UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE GRADO

RECOMENDACIÓN DE NOTICIAS EN LÍNEA BASADO EN TWITTER

ADRIÁN OTERO RODRÍGUEZ

Tutor: ALEJANDRO BELLOGÍN KOUKI

Ponente: PABLO CASTELLS AZPILICUETA

JULIO 2014

Resumen del TFG

Las redes sociales, desde su aparición en la red en la década de los 90s, han experimentado un gran crecimiento gracias a la masificación de Internet. Estas han sido objeto de estudio por parte de todo tipo de disciplinas y son un pilar dentro de la comunicación global, contando algunas de ellas con cientos de millones de usuarios registrados. Dentro de las más populares se encuentra Twitter, un servicio de microblogging con sede en Estados Unidos cuya principal característica son sus mensajes de texto limitados a 140 caracteres.

La gran cantidad de información presente en estas redes hace de su recopilación y tratamiento una tarea compleja donde, a su vez, la correcta realización de dichas tareas permite generar datos útiles en diversos ámbitos, desde capturar las preferencias comerciales actuales (tecnología, música, deportes, etc.) de distintos perfiles sociales, a opiniones políticas, tendencias o temas del momento, por ejemplo.

Dentro de la temática de tendencias o temas del momento, es especialmente relevante Twitter, donde surgieron términos como ***Trending Topic*** (para referirse a las palabras o frases más repetidas en un momento concreto), ***favoritos*** (mediante esta acción un usuario guarda sus tweets preferidos) y ***retweets*** (esta acción permite a un usuario compartir fácilmente un tweet con sus seguidores).

A partir de estos conceptos surge la posibilidad de, basándose en el número de retweets y favoritos asociados a un determinado tweet, diseñar un sistema de recomendación que utilice esta información, es decir, un sistema de filtrado de información el cual organiza una serie de ítems en un ranking ordenado intentando predecir la importancia que el usuario daría a cada uno de estos ítems. De cada tweet se pueden extraer sus enlaces a distintas noticias procedentes de webs externas a Twitter y establecer un ranking de dichas noticias a partir de los favoritos/retweets totales obtenidos en todos los tweets en los que se incluya dicho enlace.

Se pueden desarrollar distintas posibilidades para validar y medir la eficacia de un recomendador; una buena forma es estudiar los resultados del mismo en un entorno real. La plataforma Plista nos permite incluir nuestro recomendador en su sistema, generando peticiones de recomendación y devolviendo los resultados a usuarios reales. A partir de las recomendaciones realizadas, Plista ofrece la información necesaria para sacar las pertinentes conclusiones acerca de nuestro sistema de recomendación.

En este Trabajo de Fin de Grado, he propuesto, implementado, y evaluado un recomendador basado en los conceptos anteriores de popularidad en Twitter. Además, lo he integrado en la plataforma Plista, solventando restricciones impuestas por esta (tiempos de respuesta y sincronización, principalmente). A partir de las conclusiones extraídas de estos resultados se pueden establecer las bases de futuros sistemas de recomendación basados en Twitter u otras redes sociales.

Palabras Clave

Twitter, recomendador, noticias, popularidad, tiempo real, retweet, favorito.

Abstract

Social networks, since their appearance on the Web in the 90´s, have suffered an exuberant growth through the massification of Internet. They have been studied in many disciplines and, nowadays, they are an important support in global communication, having most of them with millions of registered users. The most popular social network could be Twitter, it is a microblogging service with headquarters in the U.S and its basic feature is that its text messages are limited to 140 characters.

The huge amount of information available in these networks makes its collection and processing a complex task where, on the other hand, a successful accomplishment of these tasks let generate useful data from various fields: current trade preferences (technology, music, sports, etc.), different social profiles, political opinions, trends or current issues, etc.

Related to the topic of trends or current issues Twitter is especially relevant since in this social network appeared concepts like ***Trending Topic*** (referring to words or phrases most repeated at a particular moment), ***favourites*** (the action when a user keeps her/his preferred tweets), and ***retweets*** (the action when a user shares a tweet with her/his followers).

From these concepts, the possibility of designing a recommender system based on retweets and favourites associated with a particular tweet appears. That is, an information filtering system that generates an ordered ranking of items, aiming to predict the importance that the user would give to each one of these items. From each tweet, such a system can extract its links about different news items which may come from external websites – not only from Twitter – and then establish a ranking of these news links, based, for instance, on all favourite/retweet information obtained in the tweets where these links have been included.

In this way, different possibilities can be developed to validate and measure the effectiveness of the recommender, a good way is to study the results in a real context. Plista is a platform which gives us the ability to include our recommender in their system, firing the

recommendation requests and returning the results to real users. From these generated recommendations, Plista gives us the necessary information to generate appropriate conclusions about our recommendation system.

In this Graduation Project, I have proposed, implemented, and evaluated a recommender based on the above concepts of popularity in Twitter. Besides, I have integrated it into the Plista platform, working out several restrictions imposed by it (mainly, response time and synchronization issues). From the conclusions of this project, it should be possible to establish the basis of future recommender systems based on Twitter and other social networks.

Keywords

Twitter, recommender, news, popularity, real time, retweet, favourite.

Índice de contenidos

Índice de tablas

Índice de figuras

Glosario

ACID (*Atomicity, Consistency, Isolation and Durability*): Es el conjunto de características que determinan cuándo un conjunto de instrucciones forman una transacción, estas características son las siguientes: atomicidad, consistencia, aislamiento y durabilidad.

API (*Application Programming Interface*): Una API es un conjunto de servicios o funciones que son puestos a disposición de terceros por una determinada biblioteca de forma segura tras agregarles una capa de abstracción.

Base de datos relacional: Es una base de datos basada en el modelo relacional, cuya principal característica es el uso de "relaciones", donde cada relación se representa como una tabla compuesta por registros y columnas.

Favorito (FAV): Dentro del ámbito de Twitter, un usuario puede guardar sus tweets preferidos para poder acceder a dicho tweet más tarde, de esta forma esos tweets son añadidos a una lista de tweets favoritos.

Handler: En el ámbito de Twitter, es el nombre que hace referencia a un usuario de Twitter. Se puede identificar por ir precedido del símbolo **"@"**.

Hashtag (#): En el ámbito de Twitter, es como se conoce a las etiquetas usadas para marcar palabras clave o temas dentro de un tweet. Los hashtags se pueden diferenciar por ir marcados con el símbolo **"#"** al comienzo del mismo.

HTML (*HyperText Markup Language*): Es el lenguaje utilizado en el desarrollo de páginas web. Se compone de una serie de etiquetas que son interpretadas para los elementos que componen una página web.

HTTP (*Hypertext Transfer Protocol*): Es un protocolo de red encargado de la transferencia de información entre los servidores y los clientes en la ***World Wide Web*** (conocida comúnmente simplemente como ***la web***).

IDE (***Integrated Development Environment*):** Es como se conoce al conjunto de herramientas (editor de código, compilador, depurador y constructor de interfaz gráfica) empaquetadas para su utilización en el desarrollo de aplicaciones.

JSON (*JavaScript Object Notation*): Es un formato utilizado en el intercambio de datos. JSON destaca por ser ligero y sencillo lo que lo ha convertido en una alternativa a XML.

Librería: Es una colección de funciones con un propósito bien definido. Estas bibliotecas son diseñadas de forma que sean fácilmente integrables en otros programas.

OAuth (*Open Authorization*): Es un protocolo abierto que facilita el acceso a datos protegidos a la vez que impide comprometer las credenciales de un usuario.

ORP (*Open Recommendation Platform*[1]): Plista proporciona una API encargada de proporcionar a otras aplicaciones acceso a su plataforma, mediante ORP se imponen las restricciones y estructuras de datos que se utilizarán para ello.

Plista: Es una plataforma encargada de la distribución de contenidos basada en el análisis de información con el objetivo de ofrecer al usuario información relevante.

Protocolos TCP/IP (*Transmission Control Protocol/Internet Protocol*): Es como se conoce generalmente a la familia de protocolos de internet. Consisten en un conjunto de protocolos de red que permiten la transmisión de datos entre computadoras fuera de la misma red.

Query: Este término proviene del inglés (consulta) y hace referencia a una búsqueda de información en una base de datos.

Retweet (RT): Dentro del ámbito de Twitter, es la acción por la cual un usuario publica una réplica del tweet de otro usuario, permitiéndole compartir rápidamente ese tweet con sus seguidores.

Twitter: Es una de las redes sociales más populares y está presente a nivel mundial. Esta red social permite enviar tweets, es decir, mensajes de texto plano de hasta 140 caracteres.

Seguidores: Dentro del ámbito de Twitter son usuarios que se suscriben para recibir los tweets de otro usuario.

Transacción (Bases de datos): Es la agrupación de un conjunto de instrucciones enviados a la base de datos de tal forma que se ejecutan de forma indivisible: o se ejecutan todas las instrucciones que forman la transacción o ninguna de ellas.

XML (*eXtensible Markup Language*): Es un lenguaje definido por una serie de etiquetas y utilizado en el almacenamiento de datos de forma legible. Destaca su importancia en el intercambio de información estructurada entre plataformas, donde XML es un estándar.

[1] **http://orp.plista.com/**

1. Introducción

1.1 Motivación

Las redes sociales tienen una gran influencia sobre la sociedad actual, y su reciente auge las ha convertido en una inagotable fuente de información. Esta información incluye intereses comerciales de distintos perfiles de usuario, opiniones políticas y sociales, noticias y temas del momento, gustos musicales, deportivos, literarios, etc. Recopilar, almacenar y clasificar esta información correctamente es algo fundamental que permitirá mejorar los productos ofrecidos por empresas, elaborar recomendaciones de diversos ámbitos, estudiar los distintos movimientos sociales e innumerables aplicaciones más.

Una de las principales redes sociales es Twitter[2], la segunda mayor, por detrás de Facebook[3]. Twitter fue creada en 2006 y cuenta con más de 500 millones de usuarios registrados, los cuales generan más de 65 millones de tweets y 800.000 peticiones de búsqueda al día. El elemento principal de Twitter son los tweets: mensajes de texto plano escrito por los usuarios, con la restricción de tener como máximo 140 caracteres; estos mensajes pueden incluir enlaces a otras webs, urls, imágenes, etiquetas para identificarlos (*hashtag*s), etc. Cuando un usuario publica un tweet los demás usuarios pueden compartir (*retweet*) o marcar como favorito dicho tweet, lo que permite medir la popularidad de ciertos temas y noticias de actualidad, así como el alcance o la viralidad de los usuarios y de lo que comparten [14].

Con la llegada de Internet y su rápida expansión, los medios de comunicación tradicionales se han tenido que adaptar para formar parte de este cambio, lo que se ha traducido en la presencia de estos medios en la red mediante sus propias páginas web. El objetivo de estos medios es captar la atención del usuario ofreciéndole contenido que le interese. En este contexto surgen diversos sistemas de recomendación de noticias que

[2] **https://twitter.com/**

[3] **https://www.facebook.com/**

elaboran un ranking en función de las noticias disponibles, con el objetivo principal de intentar predecir cuales resultarán más interesantes para el usuario [18, 19]. Uno de estos sistemas es Plista[4], cuya plataforma ofrece recomendaciones a diversas webs de noticias (periódicos y blogs, principalmente) con la particularidad de que permite a los desarrolladores incluir sus propias recomendaciones cuando sean necesarias.

El auge de ambos fenómenos (expansión de las redes sociales y aparición de los medios de comunicación tradicionales en Internet) abre la posibilidad de desarrollar un proyecto basado en la fusión de ambos en un entorno común, donde mediante la información obtenida de una importante red social, como es Twitter, se desarrolle un sistema de recomendación de noticias con el objetivo de ser integrado en una plataforma real de recomendación como Plista.

Aunque cada vez se desarrollan un mayor número de proyectos enfocados a la recolección, tratamiento y estudio de información extraída procedente de redes sociales, el uso de esta información enfocada a realizar recomendaciones basadas en las urls que se incluyen en muchos de los tweets no es algo frecuente, y la integración de un proyecto basado en dicha idea en una plataforma como Plista resulta novedoso en estos momentos.

A partir de los resultados obtenidos en un entorno real como es Plista se podrán elaborar una serie de conclusiones que podrían servir como punto de partida en futuros motores de recomendación basados en esta u otras redes sociales como Facebook, Google + y Tumblr.

1.2 Objetivos

El proyecto estará enfocado en alcanzar cinco objetivos perfectamente diferenciados, donde los tres primeros están relacionados con el desarrollo del sistema, mientras que los dos últimos están relacionados con el estudio de los resultados obtenidos tras el desarrollo del sistema.

[4] **https://www.plista.com/es**

1. Desarrollar un sistema de extracción de datos de Twitter para su posterior tratamiento, almacenamiento y estudio.
2. Elaborar una aplicación capaz de tratar la abundante información procedente de Twitter y garantizar su correcto almacenamiento.
3. Diseñar un sistema capaz de elaborar recomendaciones con dicha información y atender las peticiones recibidas de la plataforma Plista.
4. Realizar un estudio de los resultados obtenidos por el recomendador en un sistema real, como es Plista, y medir la calidad de las recomendaciones realizadas a los usuarios.
5. A partir de las conclusiones extraídas tras analizar los resultados, sugerir posibles modificaciones o nuevos sistemas de recomendación de noticias.

1.3 Estructura

La memoria está estructurada en ocho capítulos seguidos de varios anexos técnicos, a continuación se detalla el contenido de cada capítulo:

- **Capítulo 1. Introducción**

  Definición de la motivación del proyecto, los objetivos que busca alcanzar y estructuración de la memoria del proyecto.
- **Capítulo 2. Tecnologías a utilizar**

  Descripción de las tecnologías empleadas o tenidas en cuenta durante la realización del proyecto así como de los aspectos de dichas tecnologías que han implicado elección de estas y no otras.
- **Capítulo 3. Análisis de requisitos**

  En el análisis de requisitos se especificará el comportamiento del sistema, esto incluye tanto las capacidades que debe satisfacer como todas las restricciones en su funcionalidad.
- **Capítulo 4. Diseño y desarrollo**

  Especificación de la arquitectura de la aplicación y explicación acerca del funcionamiento de dicha aplicación.

- **Capítulo 5. Pruebas**

  Definición de los conjuntos de pruebas a los que ha sido sometida la aplicación para verificar su correcto funcionamiento y su rendimiento.

- **Capítulo 6. Resultados**

  Especificación y análisis de los resultados obtenidos en el proyecto tras la finalización del mismo.

- **Capítulo 7. Conclusiones y Trabajo Futuro**

  Análisis del proyecto, de los resultados obtenidos tras la finalización del mismo, de los conocimientos adquiridos y posibles líneas de desarrollo o cambios futuros.

- **Capítulo 8. Referencias**

  Definición de las fuentes de información utilizadas en la elaboración de este trabajo de fin de grado.

- **Anexos Técnicos.**

  En esta sección se incluirá la información complementaria que no tiene cabida en el cuerpo del documento, incluyendo ejemplos de código y de las pruebas realizadas.

2. Tecnologías a utilizar

En esta apartado se analizan las tecnologías que servirán como punto de partida en el desarrollo del proyecto. Por un lado se desarrollarán las tecnologías enfocadas en la extracción de datos desde Twitter y Plista, y por otro lado se tratarán las herramientas enfocadas al desarrollo de la aplicación.

2.1 Extracción de datos

Una de las partes fundamentales en el proyecto es la extracción de los datos necesarios desde las distintas plataformas, por un lado Twitter y por otro Plista. La interacción con ambos plataformas se realiza a través de una API proporcionada por dichas plataformas, es decir, ofrecen un conjunto de funciones y métodos para ser utilizados por otro software agregando una capa de abstracción sobre los mismos.

2.1.1 API de Twitter

Twitter pone a disposición de los usuarios tres APIs diferentes [1], cada una de las cuales representan una faceta de Twitter y permite a los desarrolladores construir sus propias aplicaciones. Estas APIs podrían ser usadas de forma combinada según las necesidades de cada aplicación.

2.1.1.1 Search API

La API de búsqueda [15] está diseñada con el objetivo de permitir al usuario realizar consultas sobre el contenido de Twitter, esto incluye desde la búsqueda de tweets aplicando diversos filtros (búsqueda de palabras clave, tweets pertenecientes a un usuario, tweets que hagan referencia a un usuario específico, en función del idioma, en función de la ubicación, etc.). También proporciona acceso a las tendencias del momento. El contenido al que se tiene acceso mediante la API de búsqueda tiene una limitación temporal de 7 días.

2.1.1.2 REST API

La REST API [9] permite a los desarrolladores acceder a funciones primitivas de Twitter como son líneas de tiempo, actualizaciones de estado y la información del usuario. Todas las operaciones que pueden realizarse a través de la web de Twitter pueden realizarse mediante esta API, esto incluye funciones como crear y publicar tweets, marcar ciertos tweets como favoritos, hacer retweet de ciertos tweets, etc.

2.1.1.3 Streaming API

Esta API interactúa con Twitter en tiempo real proporcionando gran cantidad de datos a los desarrolladores [11]. Para ello se establece una conexión HTTP permanente entre el usuario y los servidores de Twitter, la velocidad de recepción fluctuará en función del ancho de banda de ambos extremos de la conexión y de la sobrecarga en los servidores de Twitter.

2.1.1.4 Diferencias entre las distintas APIs

Dentro de estas tres APIs se pueden distinguir dos grupos, por un lado, la Search API y la REST API, cuyo funcionamiento es el de una API web, es decir, por medio de HTTP permiten acceder mediante una url a distintos contenidos, los cuales son devueltos en formatos como XML, JSON, HTML, etc. En la ***sección 2.2 "Herramientas Software"*** se desarrollan algunos de ellos.

Un ejemplo acerca del funcionamiento de dichas APIs corresponde al siguiente caso, cuyo desarrollo se muestra en la Ilustración 1: Un usuario realiza una petición a una aplicación web (1); la aplicación web recibe la petición y la envía a la API de Twitter (2); la API de Twitter recibe la petición y la responde, devolviendo el resultado a la aplicación web (3); la aplicación web recibe la respuesta, trata la información y envía la respuesta al usuario (4); el usuario recibe la respuesta a su petición inicial (5).

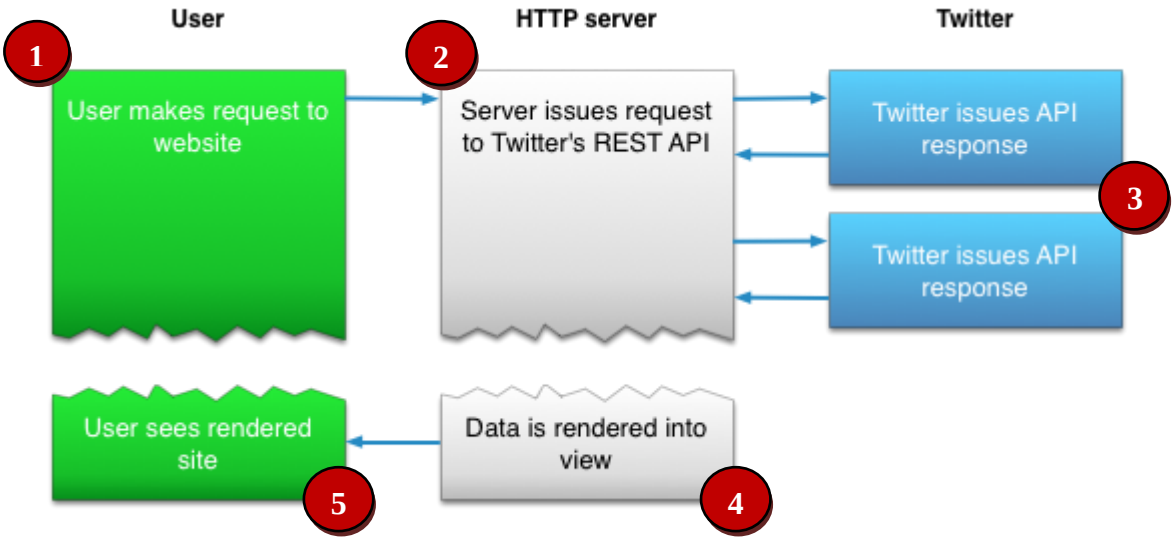

Ilustración 1. Funcionamiento de una REST API

Fuente: {"https://dev.twitter.com/sites/default/files/images_documentation/streaming-intro-1_1.png"}

Por otro lado la Streaming API no es capaz de dar respuesta a las peticiones de un usuario, si no que requiere una conexión permanente y permite establecer un flujo constante de información. Tras establecer la conexión el usuario puede llevar a cabo distintos tipos de filtrado sobre los datos recibidos según sus necesidades.

Un ejemplo del funcionamiento de dicha API corresponde al siguiente caso, cuyo desarrollo se muestra ahora en la Ilustración 2: El servidor inicia la comunicación, para lo que debe enviar una petición a Twitter (1); Twitter acepta la petición de conexión (2); se establece la conexión y comienza el flujo de datos entre Twitter y el servidor (3); en un momento dado un usuario realiza una petición contra los datos recibidos, en esta petición puede aplicar diversos filtros (4); el servidor procesa la petición y genera los resultados a partir de los datos almacenados devolviendo estos resultados al usuario (5); cuando lo desee el servidor puede detener la comunicación con Twitter (6).

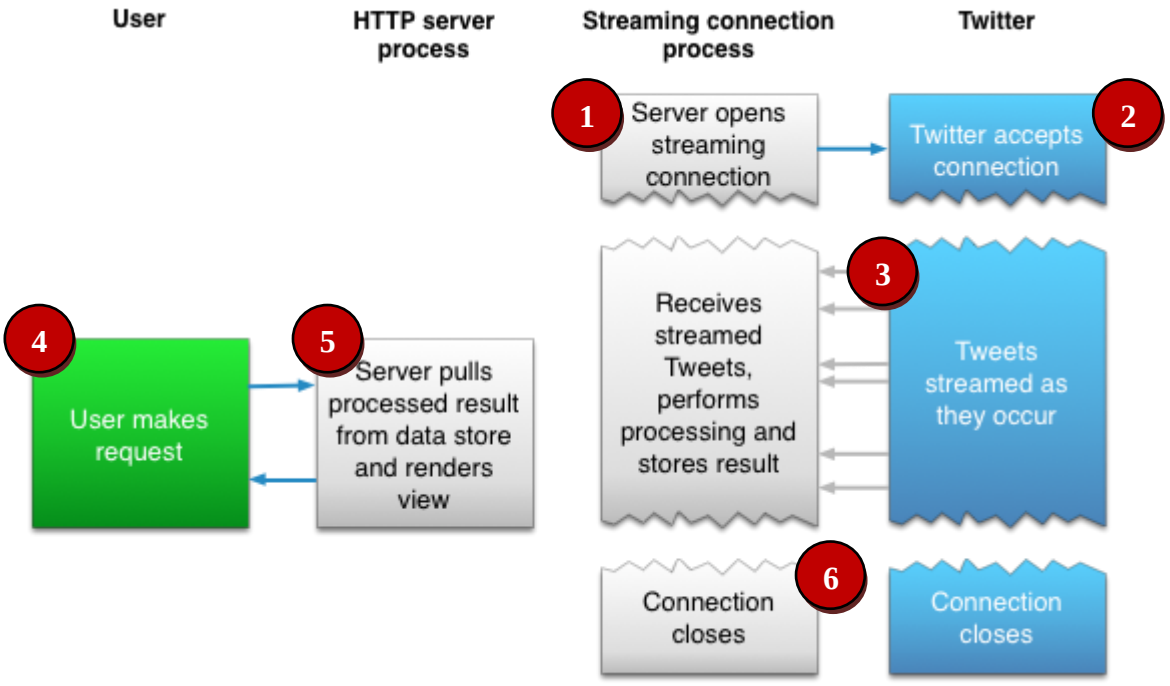

Ilustración 2. Funcionamiento de una Streaming API

Fuente: {"https://dev.twitter.com/sites/default/files/images_documentation/streaming-intro-2_1.png"}

Otra de las diferencias entre las tres APIs es la persistencia de los datos a los que da acceso Twitter. Mientras que la Streaming API entrega la información en el momento que se genera, la Search API y la REST API permiten acceder a tweets generados en el pasado. En la siguiente tabla se muestra la limitación temporal de los datos a los que Twitter da acceso en función de cada API:

	Limitación Temporal
Streaming API	**Tiempo Real**
Search API	**Últimos 7 días**
REST API	**Sin limitación temporal**

Tabla 1. Persistencia de los datos según cada API

Dependiendo del tipo de operación se puede requerir autentificación, con el mismo criterio que el acceso web.

2.1.1.5 Elección de la API para interactuar con Twitter

Debido a sus características, la API escogida es la Search API, puesto que es la que más se ajusta a las necesidades del proyecto, estas características se basan en tres puntos fundamentales requeridos por el proyecto:

- **Persistencia temporal adecuada**, es necesario recolectar tweets de actualidad, para ello la Search API proporciona tweets de hasta 7 días de antigüedad mientras que las otras APIs proporciona los tweets generados en ese instante (Streaming API) o los tweets ofrecidos no tienen una limitación temporal (REST API).
- **Búsqueda de tweets en función de una serie de palabras clave**, Search API proporciona filtros de búsqueda entre los cuales se encuentra la búsqueda por palabras clave. La Streaming API también proporciona diversos filtros de búsqueda que cumplirían con los requisitos de la aplicación, sin embargo, la REST API no está diseñada con dicha finalidad como prioridad.
- **Volumen de información**, en este punto cabe distinguir la Streaming API del resto puesto que en comparación genera un volumen de información muy superior, gran parte de ella innecesaria, pues los filtros de búsqueda se aplican tras recibir los datos. La cantidad de información a la que facilita el acceso Twitter implica un gran volumen de datos, incluso aplicando filtros de búsqueda, por lo que la Search API y REST API se ajustan mejor a las necesidades en cuanto al aprovechamiento de los recursos físicos necesarios.

En resumen, la Search API es la única que cumple con todos los requisitos del proyecto: persistencia temporal, búsqueda de tweets basada en palabras clave y el mínimo volumen de información necesario. Por ello de las tres APIs proporcionadas por Twitter es la idónea para la elaboración del proyecto.

2.1.1.6 Librería Twitter4J

La librería **Twitter4J**[5] es una librería de Java que permite integrar una aplicación con la API de Twitter de una forma sencilla [12]. Esta librería da acceso a las funcionalidades de la Search API de Twitter, entre las que se encuentra la búsqueda de Tweets relacionados con una serie de palabras clave.

Para realizar las operaciones de extracción de información desde la API de Twitter a nuestra aplicación es necesaria la autenticación en dicha API, por ello la librería Twitter4J da soporte al estándar OAuth (*Open Authorization*), un protocolo abierto que implementa un mecanismo seguro de identificación, permite a un tercero acceder a los datos de un usuario sin tener acceso a la contraseña de este. Puesto que nuestra aplicación no requiere acceso a las cuentas de los distintos usuarios de Twitter, únicamente será necesario crear una cuenta en Twitter y registrar la aplicación en la misma, garantizando el acceso a los datos públicos de Twitter a los que tiene acceso cualquier usuario estándar.

En el ***Anexo A*** ***"Proceso de autentificación en la API de Twitter"*** se detalla el proceso por el que se registra la aplicación y se integra dicho proceso de autentificación en el código de una aplicación a través del protocolo OAuth soportado por Twitter4J.

2.1.2 API de Plista

Plista permite a los desarrolladores testear sus algoritmos de recomendación en un entorno real, ofreciendo diversas estadísticas y otra información de interés [16]. Esto incluye el número de peticiones desde la plataforma al recomendador, la cantidad de "clics" de los usuarios, es decir, el número de veces que los usuarios que requieren una recomendación muestran interés por los ítems ofrecidos y acceden al enlace, el ratio entre peticiones y "clics", etc.

Plista actúa como plataforma intermediaria entre los desarrolladores y el usuario final, a continuación se describe el proceso hasta que el usuario recibe la recomendación procedente de un desarrollador desde el punto de vista de Plista: (1) El usuario accede a una web de noticias, (2) la web de noticias pide a Plista una recomendación de noticias para ese usuario,

[5] **http://twitter4j.org**

(3) Plista genera una serie de recomendaciones o (4) reenvía dicha petición a alguno de los recomendadores registrados por un desarrollador, (5) Plista devuelve a la web las noticias sugeridas para ese usuario, (6) la web de noticias muestra las sugerencias al usuario y (7) Plista recolecta una serie de información del usuario, por ejemplo si el usuario acepta alguna de las recomendaciones y accede a dicha noticia. A su vez, envía esta información a los desarrolladores. En el gráfico mostrado a continuación se detalla este proceso:

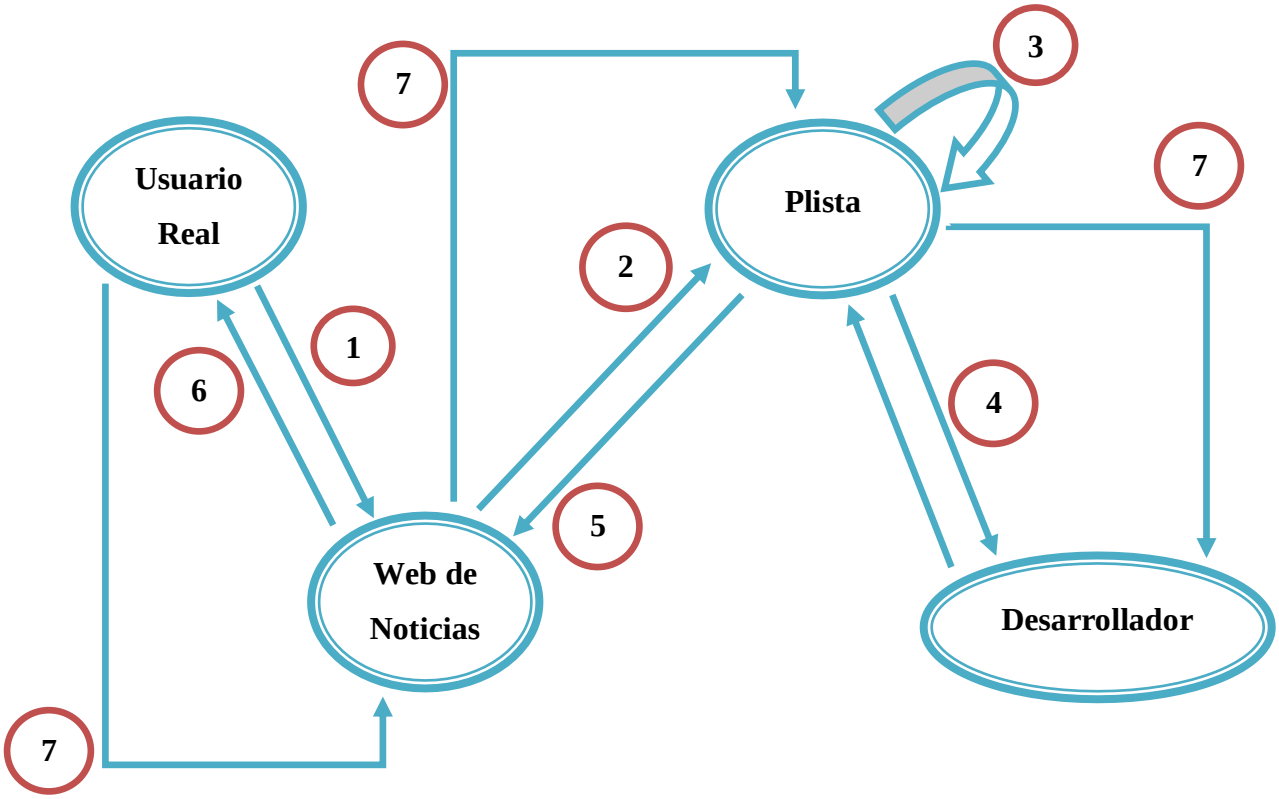

Ilustración 3. Interacción entre usuario, web, Plista y desarrollador.

La interacción entre Plista y la aplicación Java se integrará mediante el protocolo ORP, *Open Recommendation Platform*, este protocolo actúa de intermediario entre los proveedores de recomendaciones y los consumidores permitiendo a los desarrolladores realizar un seguimiento de los resultados de sus algoritmos de recomendación [17].

Este protocolo impone una serie de restricciones, las cuales un desarrollador debe tener en cuenta a la hora de implementar su aplicación, estas restricciones abarcan dos puntos clave en cuanto al rendimiento mínimo:

- **Tiempo de respuesta crítico.** El sistema debe ser capaz de responder las peticiones de los usuarios en un tiempo inferior a 100ms. El objetivo de este tiempo es garantizar la fluidez visual con el objetivo de permitir una experiencia de navegación óptima para el usuario.
- **Gran volumen de información.** El sistema debe ser capaz de gestionar grandes volúmenes de información, esto puede incluir varios miles de peticiones de recomendación por segundo.

En el caso de que la aplicación presente problemas de rendimiento y no cumpla con lo exigido dicho protocolo contempla dos posibles soluciones, o bien la relajación de las restricciones, rebajando el volumen de peticiones, o la desactivación del algoritmo implementado por el desarrollador.

A la hora de integrar dicho protocolo en una aplicación Java se utilizará la API proporcionada por Plista, en ella se deben tener en cuenta una serie de especificaciones establecidas para el correcto funcionamiento de la misma. Estas especificaciones están detalladas en los apartados situados a continuación.

2.1.2.1 Tipos de datos

En Plista se pueden distinguir tres tipos de estructuras de datos: **1) Vectores**; **2) Secuencias de vectores**, **3) Items**. Estos son todos los datos necesarios en el protocolo ORP.

- **Vectores**: Esta es la estructura de datos más sencilla y tienen asociada una ID numérica. Dentro de los vectores se pueden distinguir dos clases perfectamente diferenciadas, por un lado los vectores de entrada y por otro los vectores de salida:

    - **Vectores de entrada:** Son utilizados para describir el contexto de mensajes y eventos. Se reciben por la aplicación a través de la API de Plista. Estos vectores a su vez se dividen en los siguientes tipos:

Tipo de vector	Tipo de valor
Simple	**Un único valor (un número entero real o un ID numérico que hace referencia a una cadena)**
Lista	**Una lista de valores (identificadores numéricos nuevamente)**
Cluster	**Un mapa de valores (identificadores) u otros valores numéricos (por ejemplo puntuaciones de probabilidad)**

Tabla 2. Tipos de vectores de entrada Plista.

- **Vectores de salida:** Son siempre de tipo escalar y se utilizan para transportar la información de salida, es decir, las sugerencias de recomendación enviadas desde la aplicación a la API de Plista. Estos vectores a su vez se dividen en los siguientes tipos:

Tipo de vector	Tipo de valor
Enteros	**Un único valor entero. Utilizado para los Ids numéricos principalmente**
Reales	**Un único valor en coma flotante. Usado por ejemplo para indicar el valor de las puntuaciones**
Cadenas	**Una cadena de texto**

Tabla 3. Tipos de vectores de salida Plista.

- **Secuencias de vectores:** Los datos al ser transmitidos deben ser agrupados en función de su tipo y empaquetados en un mapa de valores donde la clave es el ID del vector y su valor es el asociado a dicho ID. Tras esta primera agrupación son agrupados nuevamente en función de su clase (vectores de salida o entrada) formándose así la secuencia de vectores. Un ejemplo de posible secuencia de vectores sería la siguiente:

```
{
        “simple”: {
                “1”: 100.
                “7”: 0.
                “11”: 4377123.
                “12”: 3131.
                “45”: 2.
                “16”: 0.
        }.
        “lists”: {
                “5”: [4323. 4324. 36654].
                “6”: [1314]
                “10”: [3. 4].
        }.
        “clusters”: {
                “28”: {
                        “400132”: 140.
                        “400133”: 10.
                        “400134”: 25
                }.
                “22”: {
                        “15”: 20
                }.
        }
}
```


Ilustración 4. Ejemplo de una posible secuencia de vectores.

Estas secuencias de vectores son representadas en formato JSON, este formato es el utilizado en todas las comunicaciones.

- **Items:** Los ítems son objetos que representan artículos o productos en Plista. Este tipo de datos tienen su propia estructura. En la siguiente tabla se muestran los campos que componen dicha estructura:

Atributo	Tipo de dato	Descripción
Id	**Entero**	Id del ítem. Debe ser único.
Title	**Cadena**	Título del ítem
Text	**Cadena**	Resumen del texto del ítem
Url	**Cadena**	Url del ítem
Domainid	**Entero**	Id del dominio. Un mismo dominio puede agrupar varios ítems
Img	**Cadena**	Url de la imagen asociada con el ítem
Create_at	**Cadena**	Fecha en que el ítem fue creado
Updated_at	**Cadena**	Fecha de la última modificación del ítem
Flag	**Entero**	Este campo se utilizará para indicar si un ítem es recomendable o no. Este indicador puede modificarse y pasar a ser no recomendable por ejemplo cuando un publicador elimina un ítem
Version	**Entero**	Contador utilizado para seguir los cambios realizados sobre el ítem. Se inicializa a 1 y aumenta con cada actualización del ítem
Filter	**Objeto**	Este campo no es usado actualmente

Tabla 4. Campos de la estructura Item.

En este proyecto los campos fundamentales, que serán indispensables cuando se capture un ítem, son los siguientes: **id**, **title**, **text**, **url**, **domainid** y **flag**.

2.1.2.2 Push Interface

La ***"Push Interface"*** es el medio utilizado en la transmisión de mensajes. Utiliza el protocolo HTTP y consultas a través de una url proporcionada por el asociado. Todos los mensajes son transmitidos a través del método POST del protocolo HTTP. Estos mensajes se componen de dos parámetros: tipo y cuerpo del mensaje. Existen los siguientes tipos de mensajes:

Tipo	Descripción
Recommendation_Request	Estos mensajes indican que se ha solicitado una petición de recomendación
Item_Update	Notificación sobre la actualización de un ítem
Event_Notification	Notificación sobre alguno de los eventos que pueden ocurrir. Por ejemplo impresiones o impresiones vacías
Error_Notification	Notificación de que un error ha ocurrido

Tabla 5. Plista Push Interface, tipos de mensajes.

Los tipos de mensaje que serán tratados en este proyecto son: los ***"Recommendation_Request"***, donde la aplicación desarrollada deberá identificar el dominio al que pertenece la petición para posteriormente devolver los ítems recomendados. Y los ***"Item_Update"***, que enviarán a la aplicación tanto los nuevos ítems como actualizaciones de los ya existentes en su base de datos.

En cuanto al parámetro "cuerpo", este se trata siempre de un objeto JSON codificado que representará el mensaje. Su contenido depende del tipo de mensaje.

2.2 Herramientas Software

El desarrollo del proyecto requiere una serie de herramientas software para lograr alcanzar los objetivos fijados, entre estas herramientas se incluyen las necesarias para gestionar la base de datos, gestionar el proyecto y las librerías que requiere o realizar el desarrollo del mismo.

2.2.1 JAVA

Java es un popular lenguaje de programación de propósito general, es decir, su funcionalidad abarca desde acceso a bases de datos hasta cálculos matemáticos, diseño de aplicaciones web, comunicación entre dispositivos, etc.

Una de las principales características de este lenguaje es su orientación a objetos. Esto implica que estos objetos son el elemento fundamental en el lenguaje, consisten en una estructura de datos (atributos) con su propio comportamiento definido (métodos) y con una identidad que les diferencia del resto de objetos (identificador)

La popularidad de Java hacen de este lenguaje idóneo para muchas tareas puesto que existe una gran variedad de bibliotecas para diversos tipos de propósitos, en el caso de este proyecto se han utilizado las librerías de Twitter4J para interactuar con la API de Twitter y SQLite para gestionar la base de datos. Además es un lenguaje que facilita la reutilización de código, gracias a lo cual los proyectos **plistarecs** y **plistaclient** (ambos proyectos se detallan en la ***sección 4.2.4 "Plista, integración con código ya existente"*),** desarrollados también en Java, se han podido utilizar como punto de partida a la hora de tratar con la API de Plista.

2.2.2 Netbeans

Netbeans[6] es un IDE (Entorno de Desarrollo Integrado), es decir, un conjunto de herramientas de programación empaquetadas en un mismo programa informático. Este entorno de desarrollo está compuesto por las siguientes herramientas: Editor de código, compilador, depurador y constructor de interfaz gráfica [6].

Netbeans fue desarrollado destinado a la programación en lenguaje Java principalmente y ofrece este conjunto de herramientas de manera libre y gratuita, sin restricciones en su uso. Por ello ha sido el entorno de desarrollo escogido en la elaboración del proyecto.

2.2.3 SQLITE

SQLite[7] es una biblioteca que implementa un motor de gestión de bases de datos SQL relacionales [2]. Su código es público y se permite su uso libre para cualquier tipo de propósito, comercial o privado. Sus dos principales características son las siguientes:

[6] **https://netbeans.org/**

[7] **http://www.sqlite.org/**

a. SQLite carece de un proceso servidor externo.

El motor de SQLite se integra en la aplicación, pasando a formar parte de la misma, de forma que el programa puede acceder plenamente a su funcionalidad simplemente mediante llamadas a las funciones y subrutinas proporcionadas por la biblioteca de SQLite. A diferencia de SQLite, en la mayoría de sistemas de gestión de bases de datos el motor es un proceso independiente a través del cual el programa se comunica con la base de datos, generalmente mediante un protocolo TCP/IP.

Por ello, SQLite permite reducir los tiempos de acceso a la base de datos puesto que sus llamadas a funciones para escribir/leer en la base de datos, situada en el propio disco, son más eficientes que la comunicación entre procesos, donde es necesario enviar las peticiones al servidor.

La particularidad de que SQLite carezca de un proceso servidor independiente también implica que no necesita ser instalada ni configurada, lo que facilita su puesta en marcha.

b. SQLite controla las transacciones en la base de datos.

SQLite controla todos los cambios realizados en la base de datos con el fin de garantizar la seguridad y consistencia de los datos involucrados, para lo cual debe ser capaz de deshacer las operaciones realizadas y devolver los datos a su inicial si se produce cualquier tipo de fallo durante la transacción. Entre los posibles fallos donde SQLite garantiza la consistencia de los datos también se contemplan las siguientes situaciones críticas:

- **Un fallo en el programa.**
- **Una caída del sistema operativo.**
- **Un fallo de alimentación.**

Por ello SQLite es un sistema de gestión de bases de datos Transaccional, lo que implica que debe cumplir con los criterios ACID (**A**tomicity, **C**onsistency, **I**solation and **D**urability). ACID es un conjunto de requisitos que deben cumplirse para que una serie de instrucciones sean consideradas como una transacción. Estas características son las siguientes:

- **Atomicidad:** Esta propiedad garantiza que se realicen todos los pasos de la transacción o ninguno en caso de fallo.
- **Consistencia:** Garantiza que la transacción llevará la base de datos desde un estado válido a otro también válido, es decir, que en el proceso no se romperán reglas y directrices que afecten a la integridad de la base de datos.
- **Aislamiento:** Esta propiedad garantiza la independencia entre varias transacciones, es decir, indica el instante en que los cambios realizados por una transacción serán visibles.
- **Durabilidad:** Una vez finalice una operación, los datos persistirán aunque el sistema falle.

Estas características hacen de SQLite la biblioteca idónea a la hora de gestionar la base de datos del proyecto, por garantizar las necesidades de este y por su simplicidad en cuanto a instalación y requisitos.

2.2.4 JSON

JSON[8] (*JavaScript Object Notation*) es un formato ligero utilizado en el intercambio de datos [3]. JSON gracias a su sencillez, facilidad de uso y capacidad de ser leído por cualquier lenguaje de programación, se ha convertido en una alternativa muy frecuente frente a XML, especialmente en entornos donde el flujo de datos transmitidos entre cliente y servidor es muy grande y la fuente de datos es fiable.

En JSON pueden aparecer los siguientes valores: nulo (null), numérico (entero o real), cadena de caracteres (entre comillas), booleano (con valor verdadero/falso), array (entre corchetes "[]"), objeto (entre llaves "{}"). Las siguientes figuras muestran la representación de objetos y arrays en JSON:

[8] **http://json.org/**

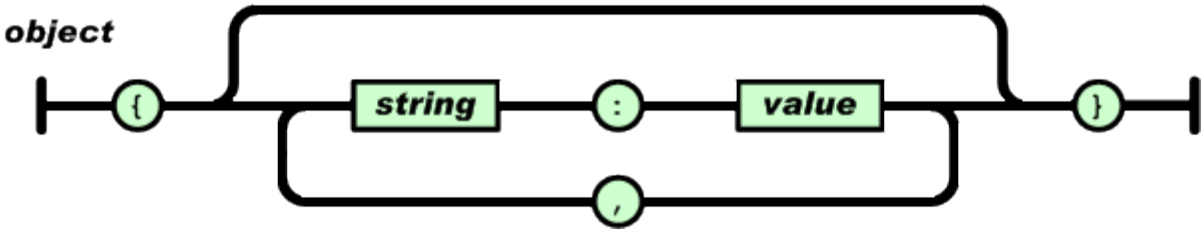

Ilustración 5. JSON, estructura objeto.

Fuente: {"http://www.json.org/json-es.html"}

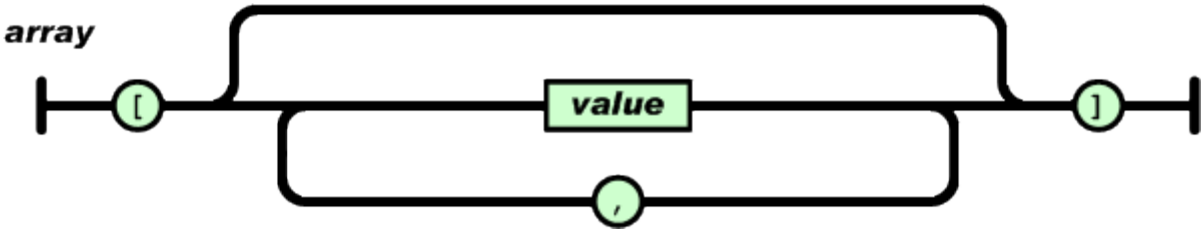

Ilustración 6. JSON, estructura array

Fuente: {"http://www.json.org/json-es.html"}

En el proyecto, JSON será el formato utilizado para recibir la información desde la API de Plista, esta información deberá ser tratada para trasladar esta información a nuestra aplicación.

2.2.5 MAVEN

Maven[9] es una herramienta utilizada en la construcción y gestión de proyectos Java cuyo objetivo es facilitar la labor del desarrollador [4]. Para definir el proyecto a construir, Maven utiliza un fichero de configuración denominado "***Project Object Model***" (**POM**), en dicho fichero se definen:

a) Los objetivos del proyecto que debe construir.
b) Sus dependencias de otros módulos y componentes externos.
c) El orden en que se construyen los elementos que componen el proyecto.

[9] **http://maven.apache.org/**

Maven permite descargar los plugins necesarios de un repositorio dinámicamente, facilitando la distribución de aplicaciones Java. Mediante el comando **mvn install**¸ Maven compila, testea (test automáticos de JUnit), empaqueta, instala y despliega el proyecto.

En este proyecto se ha escogido Maven como herramienta de gestión debido a sus características, entre ellas, y en relación a este proyecto, tienen especial relevancia las siguientes:

- Genera dentro de nuestro proyecto la estructura de directorios, ficheros fuente, ficheros de configuración etc.
- Compila, testea, empaqueta e instala el proyecto fácilmente e informa de los errores producidos en caso de que sea necesario.
- Gestiona correctamente las dependencias de nuestro proyecto con otros proyectos, permitiendo incluirlos fácilmente para utilizar las clases y métodos implementados en ellos.
- Se integra fácilmente con Netbeans [5], que ha sido la herramienta elegida para desarrollar el proyecto.

3. Análisis de Requisitos

El análisis de requisitos del proyecto facilita la comprensión de los propósitos de dicho proyecto así como la funcionalidad que debe alcanzar este. En el análisis de requisitos se especificará el comportamiento del sistema, esto incluye tanto las capacidades que debe satisfacer como todas las restricciones en su funcionalidad.

3.1 Requisitos Funcionales

Describen con detalle la funcionalidad esperada del sistema, es decir, las funciones que el sistema debe ser capaz de realizar.

- **RF1.** Debe ser capaz de realizar búsquedas en la API de Twitter utilizando palabras clave para obtener los tweets relacionados con un dominio dado.
- **RF2.** Debe poder establecer una conexión con Plista para recibir la información sobre sus ítems y peticiones de recomendación.
- **RF3.** Debe ser capaz de extraer los campos necesarios para medir la popularidad de un tweet (retweets, favoritos).
- **RF4.** Debe ser capaz de extraer las urls contenidas dentro de un tweet.
- **RF5.** Debe tener la capacidad de tratar la información recibida desde Plista para extraer el tipo de información recibida y sus campos.
- **RF6.** Debe ser capaz de almacenar la información extraída de cada uno de estos tweets en una base de datos.
- **RF7.** Debe poder incluir nuevos ítems de Plista en la base de datos.
- **RF8.** Debe poder actualizar los ítems de Plista incluidos en la base de datos si se recibe nueva información sobre ellos.
- **RF9.** Debe ser capaz de agrupar las urls almacenadas en la base de datos y organizarlas en función de su popularidad (usando retweets y favoritos).
- **RF10.** Debe ser capaz de asociar los ítems de Plista con las distintas urls obtenidas de Twitter.

- **RF11.** Debe ser capaz de atender y responder las peticiones de recomendación de Plista.

3.2 Requisitos No Funcionales

Describen los requisitos que no se refieren a la funcionalidad del sistema, sino a las propiedades. Estos requisitos imponen ciertas restricciones al sistema y son críticos, en caso de no cumplirse alguno de ellos puede suponer un problema grave.

3.2.1 Requisitos No Funcionales. API de Twitter

Los siguientes requisitos no funcionales surgen de restricciones impuestas por Twitter:

- **RNF1.** Seguridad, el sistema debe ser capaz de autenticarse en la API de Twitter usando el sistema OAuth.
- **RNF2.** Rapidez, la velocidad estará limitada a como máximo 180 consultas por cada 15 minutos contra la API de Twitter.
- **RNF3.** Tamaño, la base de datos asociada al sistema debe ser capaz de almacenar la gran cantidad de datos extraídos de Twitter.

3.2.2 Requisitos No Funcionales. API de Plista

Los siguientes requisitos no funcionales surgen de restricciones impuestas por Plista:

- **RNF4.** Rapidez, el sistema debe responder cada una de las solicitudes de recomendación en menos de 100ms.
- **RNF5.** Rapidez, el sistema debe ser capaz de tratar incluso miles de peticiones por segundo.
- **RNF6.** Tamaño, la base de datos utilizada por el sistema debe tener suficiente capacidad para almacenar todos los ítems recibidos desde Plista.
- **RNF7.** Fiabilidad, el sistema debe reducir al máximo el número de errores a la hora de realizar recomendaciones y devolver siempre ítems válidos.

4. Diseño y Desarrollo

En este proyecto se pretende diseñar un sistema de recomendación basado en información extraída de Twitter e integrar dicho sistema de recomendación en la plataforma Plista, para ello se deberán atender y dar respuesta a las peticiones de recomendación procedentes de usuarios de distintas webs de noticias dependientes de Plista.

4.1 Diseño del Proyecto

A la hora de elaborar el diseño del proyecto es especialmente relevante conocer el objetivo final del mismo y ser capaz de dividir este problema en partes más sencillas. En este proyecto es posible diferenciar **dos grandes sistemas perfectamente diferenciados**, por un lado se sitúan todos los **objetivos relacionados con Twitter** y por otro lado los **objetivos basados en la plataforma Plista**.

Esta primera división del problema tendrá como resultado dos grandes sistemas, los cuales deberán trabajar simultáneamente al terminar el desarrollo del proyecto, aunque separados el uno del otro, donde **la base de datos SQL se utilizará como nexo de unión entre ambos**.

Por un lado, dentro del **sistema basado en Twitter**, se situará la extracción de datos desde Twitter, el tratamiento de esta información y la generación del ranking de popularidad. Por otro lado, dentro del **sistema basado en Plista**, se realizarán tareas como la extracción de datos desde Plista y la gestión de las peticiones de recomendación recibidas desde dicha plataforma.

A su vez es posible dividir ambos sistemas en módulos más sencillos, diferenciados según su función dentro del proyecto:

1) **Extracción de datos desde Twitter y tratamiento de la información.**
2) **Generación del ranking con las noticias más populares en Twitter.**
3) **Extracción de datos desde Plista y tratamiento de la información.**
4) **Gestión de las peticiones de recomendación de Plista.**

En la siguiente figura se muestra una visión en conjunto del proyecto y la organización de sus sistemas y los módulos que los componen:

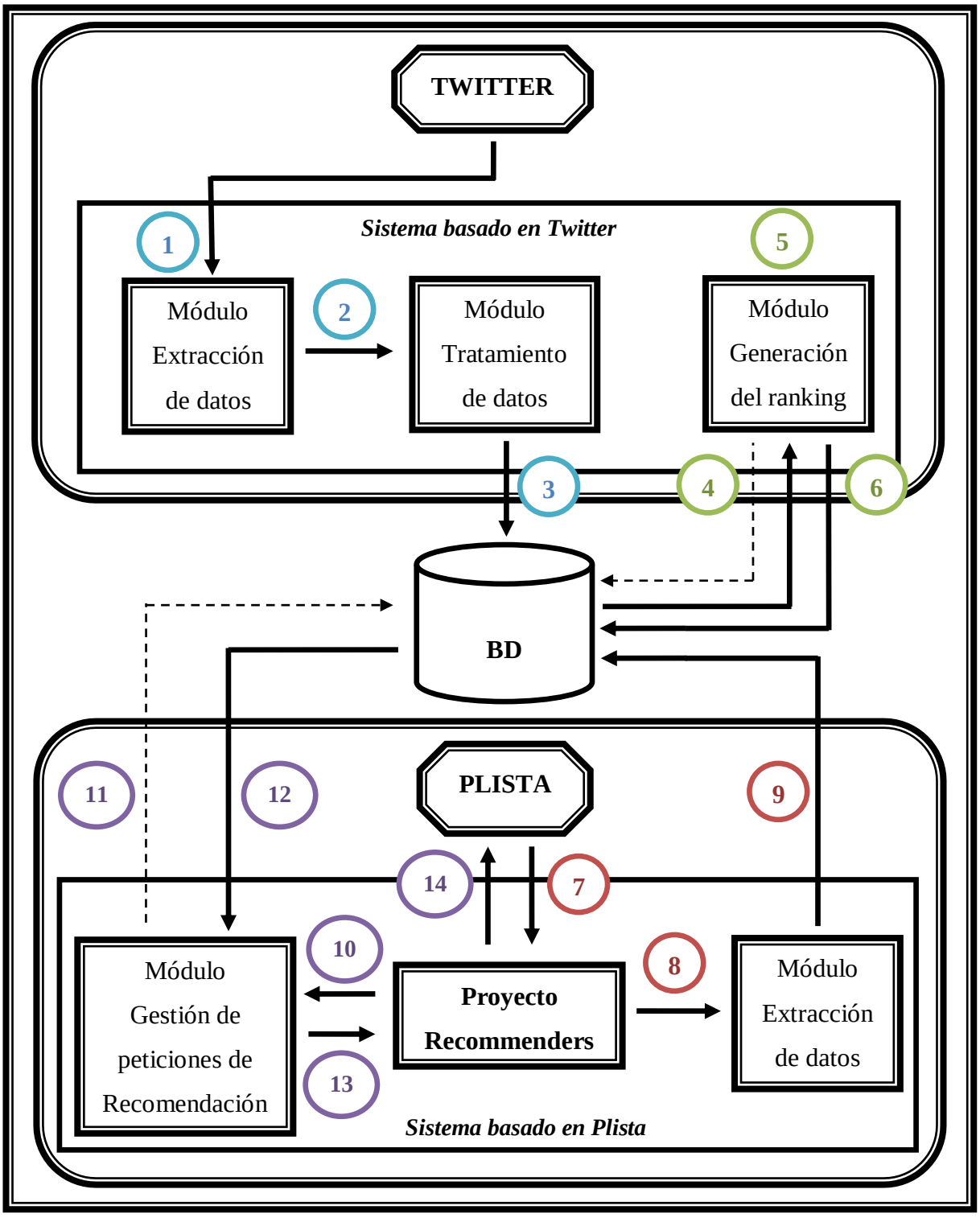

Ilustración 7. Organización del Proyecto.

En esta organización se pueden apreciar claramente diferenciados los dos sistemas que componen el proyecto así como los módulos que los componen y el uso de la base de datos SQL como nexo de unión entre ambos sistemas. A continuación se detallan cada una de las acciones representadas en la figura anterior:

- **Sistema basado en Twitter:**
  - **1)** Con una cierta frecuencia, la aplicación realiza una búsqueda en la API de Twitter y recibe desde ella los resultados de la búsqueda.
  - **2)** La información recibida es tratada, conservando los datos útiles.
  - **3)** Estos datos son almacenados en la base de datos.
  - **4)** Cada cierto tiempo, el ranking de popularidad de las noticias de Twitter se actualiza, para ello se solicita la información almacenada en la base de datos.
  - **5)** La base de datos devuelve todos los tweets almacenados y su información asociada.
  - **6)** Se agrupan las urls y se almacenan en la base de datos ordenadas en función de su popularidad.

- **Sistema basado en Plista:**
  - **7)** Desde Plista es posible recibir o bien un ítem de Plista o bien una petición de recomendación. Se utilizan los métodos implementados en los proyectos **plistarecs** y **plistaclient (detallados a continuación, en la *sección 4.2.4 "Plista, integración con código ya existente")*** para tratar la información y extraer tanto el tipo de datos como sus campos.
  - **8)** En caso de tratarse de un ítem de Plista se extraen los campos útiles.
  - **9)** Se introduce este nuevo ítem en la base de datos (o se actualiza la información del mismo si ya se encontrara en la base de datos).
  - **10)** En caso de tratarse de una solicitud de recomendación se solicitan los ítems más populares en ese momento asociados a ese dominio.
  - **11)** Los ítems más populares asociados a cada dominio se mantendrán en el programa y se actualizarán utilizando la base de datos cada cierto número de peticiones o en caso de no obtener suficientes ítems.

12) La base de datos devolverá los ítems más populares tras cruzar la información de Twitter y Plista.

13) Se devolverán los ítems recomendados y en caso de que no sean suficientes se usará el "Recent Recommender" como recomendador de apoyo para generar las recomendaciones restantes.

14) Se devolverán los ítems recomendados a la API de Plista.

4.2 Implementación del Proyecto

En esta sección se desarrollará la explicación detallada acerca de la implementación del proyecto para cada uno de los módulos descritos anteriormente.

4.2.1 Base de datos SQL

La base de datos se implementará en Java a través de la librería SQLite. La base de datos debe ser capaz de almacenar correctamente tanto la información recibida desde Twitter como la recibida desde Plista y la forma en que es posible relacionar ambos "mundos".

Inicialmente el proyecto requiere cierta información que debe ser cargada manualmente para rellenar tres tablas básicas, las cuales permitirán al proyecto comenzar a funcionar desde cero. Estas tablas contendrán información sobre:

- Los dominios de los publicadores de noticias asociados a Plista.
- Las consultas que se enviarán a Twitter para obtener tweets relacionados con noticias publicadas a través de urls asociadas a dichos publicadores.
- La correspondencia entre los identificadores de los dominios de Plista y los asociados a cada Tweet capturado.

En el ***Anexo B*** ***"Ficheros de carga inicial en la base de datos"*** se incluye la información que será introducida en la base de datos en la carga inicial del proyecto; no obstante, sirva de ejemplo el periódico alemán "***Tagesspiegel***" como uno de los dominios, para el cual usamos como consulta "***www.tagesspiegel.de***".

En las siguientes figuras se muestra la forma en que ha sido implementada la base de datos incluyendo las relaciones que permiten unir la información extraída de Twitter con los datos de Plista:

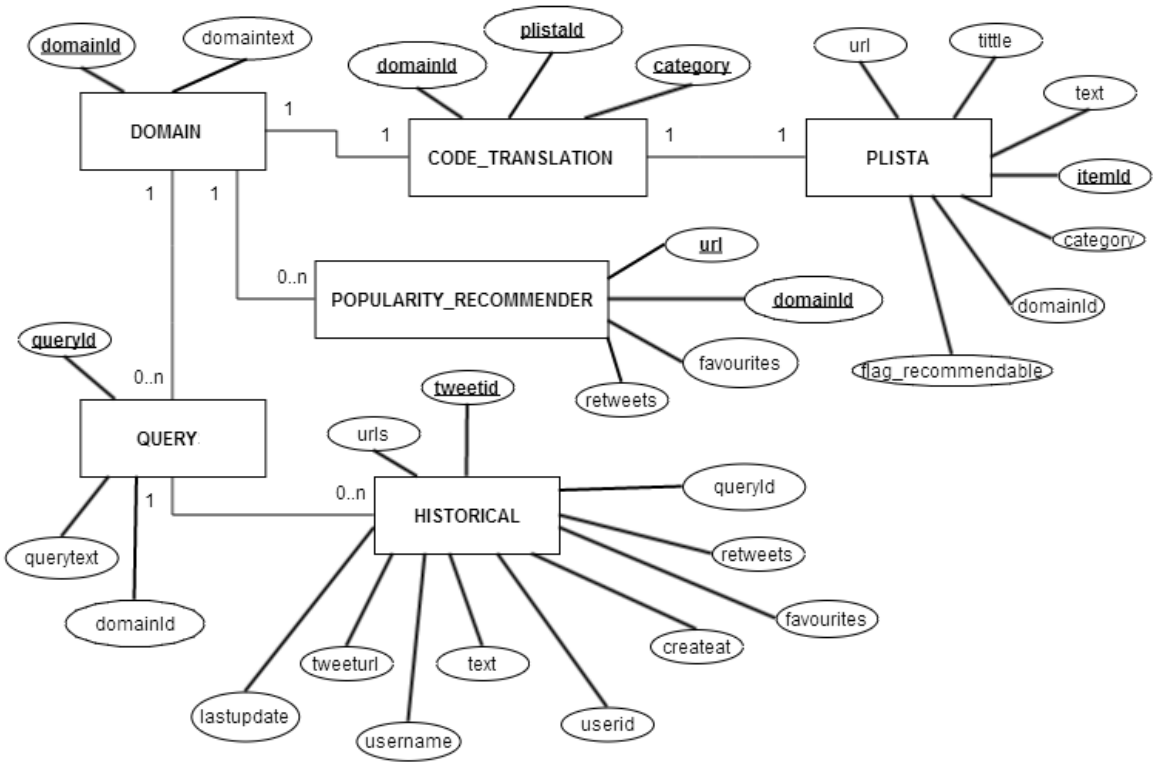

Ilustración 8. Base de datos, esquema implementado.

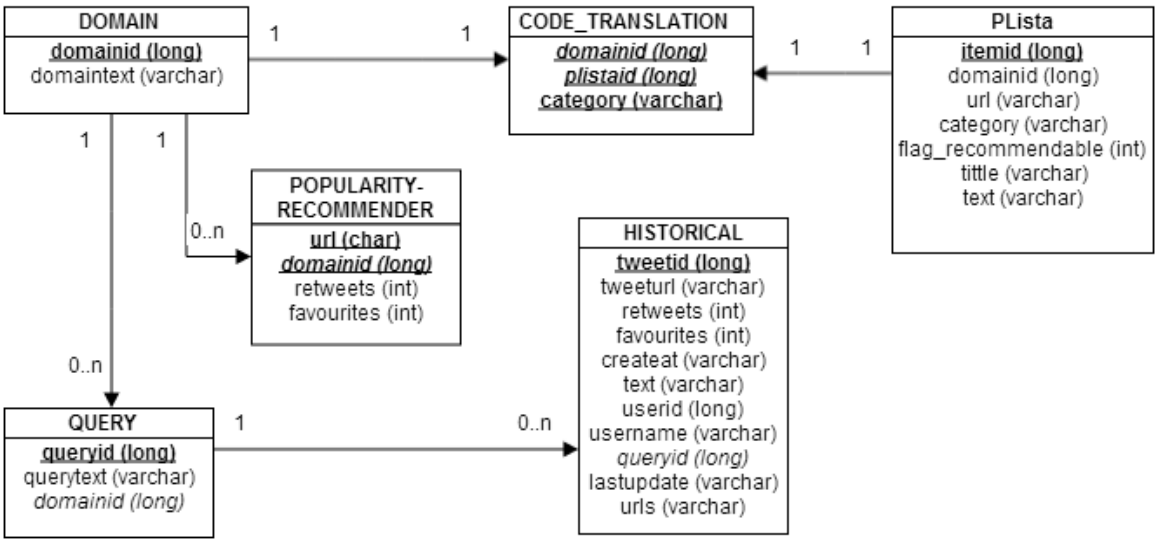

Ilustración 9. Base de datos, tipo de datos de cada campo.

En las figuras anteriores se puede observar el esquema de la base de datos, a continuación se detalla la utilidad de cada una de las tablas que la componen:

- **Domain.** En esta tabla se almacenarán el nombre de dominio de los publicadores de noticias y su identificador asociado.
- **Query.** En ella se guardarán las consultas que se lanzarán contra la API de Twitter, un identificador único para cada consulta y el identificador del dominio asociado.
- **Historical**. La información extraída de los tweets recibidos desde la API de Twitter será almacenada en esta tabla.
- **Plista**. Los campos útiles extraídos de los ítems recibidos desde la API de Plista serán almacenados en esta tabla.
- **Code_Translation**. En esta tabla se almacenan las relaciones entre los identificadores de dominio de los elementos de Plista y los de Twitter.
- **Popularity_Recommender**. En esta tabla se almacenarán las urls contenidas en los tweets agrupadas y ordenadas en función de su popularidad (usando retweets y favoritos de los tweet implicados).

En este proyecto todos los métodos relacionados con la gestión de dicha base de datos han sido implementados en una misma clase Java. En dicha clase, para cada una de las tablas se implementan los siguientes métodos:

- **Creación de la tabla en la base de datos.**
- **Inserción de elementos en la tabla.**
- **Actualización de un elemento en la tabla, utilizando la clave primaria.**
- **Búsqueda en las tablas de la base de datos.**
- **Destrucción de todo el contenido de una tabla de la base de datos.**

Además se han implementado una serie de métodos con el objetivo de realizar funciones adicionales que involucran a la base de datos [10, 20]:

- Obtener las urls incluidas en la tabla Popularity_Recommender ordenadas según retweets y favoritos, donde ambos elementos pueden tener distinto peso.

- Incluir en cada tweet almacenado en la base de datos el identificar del dominio al que pertenece la consulta (query) de la que proviene para posteriormente poder buscar tweets en función de un identificador de dominio dado.
- Refrescar la tabla Popularity_Recommender a partir de los tweets almacenados en la base de datos. Este proceso se detallará en la ***sección 4.2.3 "Generación del ranking con las noticias más populares en Twitter"***.

4.2.2 Extracción de datos desde la API de Twitter y tratamiento de la información

En el proceso de extracción de datos desde la API de Twitter se utilizará la librería **Twitter4J** ya descrita anteriormente en la ***sección 2.1.1.6 "Librería Twitter4J".*** Esta librería se encargará de hacer transparente la interacción con la API de Twitter permitiendo conectar con ella a través distintos métodos [13].

El primer paso a la hora de comenzar el desarrollo del proceso encargado de la extracción de datos desde Twitter es autenticarse en la API, para ello es necesario seguir el proceso descrito en el ***Anexo A*** **"*Proceso de autentificación en la API de Twitter"***. En este anexo se detallan tanto los pasos necesarios para obtener las claves de autenticación necesarias, como el código necesario para autenticarse por medio de la librería **Twitter4J** utilizando dichas claves.

Una vez autenticados en la API de Twitter, la librería Twitter4J pone a nuestra disposición varios métodos que nos permitirán buscar en Twitter a través de su API. A continuación se muestra la implementación de este proceso:

1) Establecer las **propiedades de configuración de Twitter4J**, esto incluye indicar las claves de acceso a la API de Twitter mediante OAuth (ver Ilustración 10).

```
//Twitter Conf.
System.setProperty("twitter4j.loggerFactory", "twitter4j.NullLoggerFactory");
twitter4j.conf.ConfigurationBuilder cb = new twitter4j.conf.ConfigurationBuilder();
cb.setDebugEnabled(DEBUG);
cb.setDebugEnabled(DEBUG).setOAuthConsumerKey("**********");
cb.setDebugEnabled(DEBUG).setOAuthConsumerSecret("**********");
cb.setDebugEnabled(DEBUG).setOAuthAccessToken("**********");
cb.setDebugEnabled(DEBUG).setOAuthAccessTokenSecret("**********");
```

Ilustración 10. Twitter4J, propiedades de configuración.

2) Obtener una instancia de la **clase *Twitter***, definida en la librería Twitter4J como se indica en la Ilustración 11; esta instancia es reutilizable.

```
TwitterFactory tf = new TwitterFactory(cb.build());
twitter = tf.getInstance();
```

Ilustración 11. Twitter4J, obtener instancia de la clase Twitter.

3) **Se crea la consulta** (o query) **que se lanzará contra Twitter**, para ello se utiliza el método getQuery() asociado a la variable query, esto devuelve una cadena correspondiente a la consulta que deseamos realizar. En el proyecto estas consultas se corresponderán con el contenido del **campo querytext de la tabla Query**.
4) Mediante un bucle se irán recorriendo todos los resultados que devuelve el proceso anterior, estos no son todos los existentes pues la API de Twitter limita los resultados.
5) **Para cada tweet se trata la información y se recogen los atributos** que serán necesarios para implementar nuestro recomendador, para almacenar dichos atributos se ha creado una clase Java (***"TweetData"***) cuyas variables coincidirán con los campos de la **tabla Historical** de nuestra base de datos.
6) Como resultado de esta consulta contra la base de datos **se devolverá una lista de tweets**, almacenados en objetos ***TweetData***. Se puede ver en la Ilustración 12 la implementación de este paso junto con los pasos anteriores 3, 4 y 5.
7) Tras almacenar en objetos los tweets capturados desde la API de Twitter, estos se introducirán en la **tabla Historical** de la base de datos. En el método implementado encargado de insertar los tweets en la base de datos destaca la captura de excepciones provocadas por una violación de la clave primaria, en cuyo caso se realiza una actualización de la información asociada almacenada (en lugar de la inserción). La Ilustración 13 muestra esta implementación.

```
try {
    Query q;
    QueryResult result;

    q = new Query(query.getQuery());

    do {
        q.count(500);
        result = twitter.search(q);
        List<Status> tweets = result.getTweets();

        TweetList = new ArrayList<TweetData>();

        for (Status tweet : tweets) {

            String urls = "";
            URLEntity[] urlEnt = tweet.getURLEntities();
            for(URLEntity url : urlEnt){
                urls = urls + url.getExpandedURL()+ "\n";
            }

            if (!tweet.isRetweet()) {

                String tweeturl = "http://twitter.com/" +
                                  tweet.getUser().getScreenName() +
                                  "/status/" + tweet.getId();

                TweetData TD = new TweetData(tweet.getId(),
                                             tweeturl,
                                             tweet.getRetweetCount(),
                                             tweet.getFavoriteCount(),
                                             tweet.getCreatedAt(),
                                             tweet.getText().replace("'","").replace("\"",""),
                                             tweet.getUser().getId(),
                                             tweet.getUser().getScreenName(),
                                             query.getQueryId(),
                                             null,
                                             urls);
                TweetList.add(TD);
            }
        }

    } while ((q = result.nextQuery()) != null);

    return TweetList;

} catch (TwitterException te) {

    System.err.println("Failed to search tweets: " + te.getMessage());
    return null;
}
```

Ilustración 12. Twitter4J, búsqueda en la API de Twitter.

```
public void Insert_Tweets(List<TweetData> TweetList){
    String sql = null;
    TweetData tweet_actual = null;

    try {
        connection.setAutoCommit(false);

        Statement statement = connection.createStatement();
        statement.setQueryTimeout(30);  // set timeout to 30 sec.

        for (TweetData tweet : TweetList) {

            tweet_actual = tweet;
            sql = "INSERT INTO HISTORICAL (TWEETID,TWEETURL,RETWEETS,FAVOURITES,CREATEAT,"
                    + "TEXT,USERID,USERNAME,QUERYID,LASTUPDATE,URLS) VALUES (" +
                    tweet.getId() + ", " +
                    "'" + tweet.getTweeturl() + "', " +
                    tweet.getRetweets() + ", " +
                    tweet.getFavoritos() + ", " +
                    "'" + tweet.getCreateat() + "', " +
                    "'" + tweet.getTexto() + "', " +
                    tweet.getUserid() + ", " +
                    "'" + tweet.getUsername() + "', " +
                    tweet.getQueryId() + ", " +
                    "'" + tweet.getLastUpdate() + "', " +
                    "'" + tweet.getUrls() + "');";
            statement.executeUpdate(sql);
        }

        statement.close();
        connection.commit();

    } catch (Exception e) {
        // Comprueba si se trata de un fallo de violacion de clave primaria
        // (En ese caso actualizar registro)
        if(e.getMessage().contains("UNIQUE constraint failed")){
            this.Update_Tweet(tweet_actual);
        } else {
            System.err.println(sql);
            System.err.println( e.getClass().getName() + ": " + e.getMessage());
        }
    }
}
```

Ilustración 13. Twitter4J, inserción de tweets en la BD.

El sistema encargado de capturar la información de Twitter lanzará estos métodos de captura e inserción de tweets constantemente, sin embargo, debido a las **restricciones i**mpuestas por Twitter a la hora de enviar peticiones de búsqueda (180 peticiones cada 15 minutos) [15], tras finalizar cada búsqueda de una de las consultas en la API de Twitter, el sistema entrará en modo de espera durante 1 minuto, tras lo cual se realizará la búsqueda de la siguiente consulta en la API de Twitter.

Una vez completado este proceso para todas las consultas, el sistema esperará durante 1 hora hasta volver a iniciar dicho proceso; esta espera se debe a que, en nuestras pruebas, actualizar la base de datos con una frecuencia mayor no aporta nueva información.

En caso de que el número de consultas se incremente en gran medida o las necesidades de la aplicación cambien, dichos parámetros de espera deberán ser reajustados.

4.2.3 Generación del ranking con las noticias más populares en Twitter

El proceso de generación/actualización del ranking con las noticias más populares se enmarca dentro del sistema encargado de capturar la información de Twitter, esta actualización del ranking se produce tras finalizar la búsqueda en la API de Twitter de cada una de las consultas almacenadas en la base de datos, justo antes de que el sistema entre en espera durante 1 hora.

El proceso de actualización del ranking de popularidad se realiza íntegramente en la base de datos, este proceso conlleva los siguientes pasos:

1) Se crea una tabla temporal, ***"temp"***, donde se almacenan: url almacenada en el tweet (***url***), id del dominio asociado (***domainid***), número de retweets (***retweets***) y número de favoritos (***favourites***). Para ello es necesario cruzar la tabla **Historical** y la tabla **Query** (para obtener el ***domainid***).
2) Partiendo de la tabla ***"temp"***, generada anteriormente, se agrupará la información en función del par: ***url + domainid***; y se calculará el total de ***retweets*** y ***favourites*** asociados a cada uno de estos pares.
3) Finalmente se insertarán estos datos en la tabla **Popularity_Recommender.**

En el ***Anexo C "Generación del ranking de popularidad (implementación)"*** se incluye el código desarrollado para implementar este proceso.

4.2.4 Plista, integración con código ya existente (Proyecto Recommenders)

Como punto de partida a la hora de integrar el proyecto con la API de Plista se han utilizado los proyectos **plistaclient**[10] y **plistarecs**[11] (para abreviar, 'Proyecto Recommenders' [7, 8]). Estos proyectos consisten en un cliente y un servidor que hacen transparente la interacción con la API de Plista al desarrollador. Concretamente se utilizarán las clases implementadas en estos proyectos para tratar los objetos JSON procedentes de los mensajes enviados por la API de Plista y se extenderán los métodos correspondientes a generar nuevas recomendaciones.

4.2.4.1 Gestión de las peticiones de recomendación en Plista

Dentro del **proyecto plistarecs** se incluye la implementación de varios tipos de recomendadores, los cuales servirán de ejemplo a la hora de desarrollar la implementación de nuestro recomendador, entre estos tipos de recomendadores se encuentran los siguientes:

- **Recent Recommender.** Este recomendador es el más sencillo de todos, su funcionamiento se basa en devolver los últimos artículos/ítems del dominio para el que se solicita la recomendación.
- **Lucene Recommender.** Este recomendador guarda un índice de todos los artículos recibidos a través de la API de Plista y en función del artículo que está siendo leído por el usuario que requiere la recomendación busca los artículos relacionados basándose en el título y el texto.
- **Category-based Recommender.** Este recomendador es similar al Lucene Recommender con la diferencia de que la búsqueda de artículos relacionados se basa en la categoría del artículo.
- **Combined Recommender.** Este recomendador se basa en la combinación de dos de los tipos de recomendadores descritos anteriormente, donde uno de ellos es el recomendador principal y otro un recomendador de apoyo. El recomendador de

[10] **https://github.com/recommenders/plistaclient**

[11] **https://github.com/recommenders/plistarecs**

apoyo actúa en los casos donde el recomendador principal no es capaz de devolver el número de ítems a recomendar solicitado.

De estos recomendadores, se utilizarán el **Combined Recommender** y el **Recent Recommender** en la implementación el proyecto. Inicialmente no habrá suficiente información almacenada en la base de datos y la aplicación será incapaz de generar la lista de recomendaciones a partir de los datos disponibles cuando Plista lo solicite, por ello se utilizará el **Combined Recommender** donde nuestro recomendador (**Twitter Recommender**) será el principal y el **Recent Recommender** actuará como apoyo. Esto implica que cuando el sistema reciba una petición de recomendación, el **Combined Recommender** llamará al **Twitter Recommender**, y en caso de que dicho recomendador no devuelva suficientes elementos se pedirá al **Recent Recommender** más recomendaciones, tantas como elementos totales se hayan requerido inicialmente, pues entre los elementos devueltos por ambos recomendadores podría haber coincidencias.

En la implementación del **Combined Recommender** es necesario extender la clase abstracta *AbstractCombinationRecommender*, la cual, a su vez, implementa la interfaz *Recommender*. A la hora de construir el **Combined Recommender** se debe pasar como argumentos ambos recomendadores (**Twitter Recommender** y **Recent Recommender**) y sobrescribir el método ***recommend***, encargado de gestionar las peticiones de recomendación.

```
public class CombinedTwitterRecommender extends AbstractCombinationRecommender implements Recommender {
    public CombinedTwitterRecommender() {
        super(new TwitterRecommender(), new RecentRecommender());
    }

    @Override
    public List<Long> recommend(Message input, Integer limit) {
        TwitterRecommender.tempLog.println("CTR:recommend_in\t" + input);
        List<Long> a = super.recommend(input, limit);
        TwitterRecommender.tempLog.println("CTR:recommend_out\t" + a);
        return a;
    }
}
```

Ilustración 14. Implementación del Combined Recommender.

En el ***Anexo D*** ***"Clase AbstractCombinationRecommender, método recommend (implementación)"*** se muestra la implementación del método ***recommend*** utilizado para gestionar ambos recomendadores, el **Twitter Recommender** como **base** y el **Recent Recommender** como **apoyo**.

En cuanto al recomendador implementado en la clase **Twitter Recommender**, su funcionamiento se basará en los siguientes pasos:

1) Se recibe la petición de recomendación mediante una llamada al método ***recommend*** implementado en esta clase. En esta petición se incluye tanto el **identificador del dominio** que solicita las recomendaciones como el **número de ítems** a recomendar que se espera recibir de vuelta.
2) Los **ítems más populares** para cada dominio se almacenarán en una **tabla hash** que se consultará para obtener rápidamente ítems a recomendar.
3) Cada 100 peticiones de recomendación o en caso de no encontrar suficientes ítems recomendados, se **actualizará** esta **tabla hash**, este proceso de actualización consistirá en:
    a. Utilizando el identificador de dominio (**domainid**) de **Plista** encontrar el **domainid** asociado a la información de **Twitter**, para lo cual es necesario consultar la **tabla Code_Translation** de la base de datos.
    b. Buscar en la base de datos las **urls** asociadas a ese dominio **ordenadas en función de su popularidad**, para ello se debe consultar la **tabla Popularity_Recommender**. Los pesos asociados a retweets y favoritos a la hora de establecer la popularidad de una url se obtendrán de un fichero de configuración.
    c. Consultar si estas url pertenecen a algún ítem de Plista, para ello es necesario **consultar** si dicha **url se encuentra** en la **tabla Plista** y consultar el campo **flag_recomendable**, el cual indica si es posible recomendar ese ítem o no.
    d. Una vez obtenidos suficientes ítems o en caso de no haber más urls asociadas a dicha dominio, aunque no sean suficientes ítems, se **devolverán los ítems recomendados**.

En el ***Anexo E*** ***"Clase TwitterRecommender, métodos de recomendación de ítems y actualización de la tabla hash (implementación)"*** se incluyen los códigos asociados a los métodos de la clase Twitter Recommender que implementan la recomendación de ítems y la actualización de los ítems almacenados en la tabla hash.

4.2.4.2 Extracción de datos desde la API de Plista y tratamiento de la información

En algunas ocasiones la API de Plista en vez de peticiones de recomendación enviará "**Item Updates**", esto consiste en la actualización de un ítem asociado a alguno de los publicadores inscritos en la plataforma Plista. Estos "**Item Updates**" pueden contener tanto nuevos ítems como actualizaciones de los ya existentes, por ejemplo un ítem recomendable (cuyo flag_recomendable tiene el valor de verdadero) pasa a ser no recomendable (a causa de que este ítem ya no esté disponible, por ejemplo).

La extracción de datos desde la API de Plista se realizará a través de la sobrescritura del método ***update***, dicho método será llamado cada vez que la aplicación reciba un "**Item Update**" y su función será la de insertar en la base de datos los nuevos ítems recibidos desde Plista y actualizar la información de los ya existentes en la base de datos.

Nótese que este tipo de peticiones son críticas para nuestro recomendador, ya que necesita conocer la url de los ítems para poder decidir si conoce su número de retweets/favoritos. Eso contrasta con otros recomendadores donde esta información (el contenido de los ítems) no es necesaria y les basta con procesar las interacciones entre los ítems de un dominio, como por ejemplo el **Recent Recommender**.

5. Pruebas

Puesto que el proyecto se divide en dos sistemas diferenciados compuestos por varios módulos, el plan de pruebas se ha estructurado también siguiendo este modelo, de forma que se han realizado las pruebas primero sobre cada módulo por separado (pruebas unitarias) y finalmente se ha probado todo el conjunto (pruebas de integración y de sistema).

5.1 Pruebas unitarias sobre el sistema basado en Twitter

En estas pruebas unitarias se comprobará el funcionamiento por separado de los módulos que forman el sistema basado en Twitter. Estas han sido las pruebas más representativas en este módulo:

I. **Pruebas sobre la clase encargada de leer los ficheros de configuración inicial.**
   a. Se comprueba que la información almacenada en el programa coincide con el contenido de dichos ficheros. El contenido de estos ficheros se detalla en el ***Anexo B*** ***"Ficheros de carga inicial de la base de datos"***.
   b. Se comprueba que en caso de que algún dato contenido en los ficheros no esté en el formato correcto no se comprometa la integridad del resto del sistema.

II. **Pruebas sobre la clase encargada de interactuar con la API de Twitter a través de la librería Twitter4J.**
   a. El módulo es capaz de acceder a la API de Twitter usando las claves definidas para autentificarse en ella. En caso de no realizarse correctamente este proceso se mostrará el siguiente mensaje de error:
   "Failed to search tweets: 401: Authentication credentials (https://dev.twitter.com/pages/auth) were missing or incorrect. Ensure that you have set valid consumer key/secret, access token/secret, and the system clock is in sync".
   b. Este módulo es capaz de capturar tweets utilizando la API de Twitter.
   c. Se comprueba que las variables asociadas a los campos de cada tweet han sido construidas correctamente. En el ***Anexo F*** ***"Pruebas, tweet almacenado en el***

sistema y tweet almacenado en la web de Twitter", se muestra gráficamente el resultado de esta prueba.

d. Utilizar una palabra de uso frecuente como consulta (con el objetivo de generar muchos resultados), verificar que el diseño del sistema impide que se violen las restricciones de tráfico impuestas por la API de Twitter.

III. Pruebas sobre la clase encargada de gestionar los procesos asociados a la base de datos.

a. El módulo inicializa correctamente la base de datos y, en caso de no existir las tablas necesarias en la base de datos, estas son creadas.
b. Comprobar que los métodos de inserción asociados a cada tabla de la base de datos se realizan satisfactoriamente.
c. Comprobar que los métodos de actualización asociados a cada tabla de la base de datos se realizan satisfactoriamente.
d. Comprobar que los métodos de lectura de registros asociados a cada tabla de la base de datos se realizan correctamente.
e. El módulo realiza la eliminación de las tablas en la base de datos con éxito.
f. Una inserción de un elemento cuya clave primaria (PK) viola la condición de ser única es detectada y evitada. Verificar que en ese caso se llama al método encargado de actualizar ese registro en la tabla.

5.2 Pruebas unitarias sobre el sistema basado en Plista

En estas pruebas unitarias se comprobará el funcionamiento por separado de los módulos que forman el sistema basado en Plista. Excepto donde se especifique lo contrario, se asume el mismo peso tanto para retweets como para favoritos a la hora de valorar la popularidad de un elemento, más específicamente, un peso de 1.0 para cada uno de ellos. Estas han sido las pruebas más representativas en este módulo:

I. Pruebas sobre el módulo encargado de gestionar las solicitudes de recomendación.

a. Este módulo es capaz de capturar los pesos asignados a retweets y favoritos a la hora de establecer su popularidad desde un fichero de configuración.

Comprobar que los pesos almacenados en el fichero y los capturados se corresponden.

b. El módulo es capaz de capturar los campos asociados a cada petición de recomendación. Los campos más relevantes son el número de ítems solicitados por la recomendación y el dominio que la solicita. Devolver una lista vacía si alguno de ellos no existe o es null.
c. Se obtienen las urls correctas, aquellas con mayor número de retweets y favoritos asociados a un dominio. Para ello verificar que los resultados coinciden con el contenido de la tabla *Popularity_Recommender*.
d. El sistema es capaz de encontrar el ítem de Plista asociado a una url dada. Comprobar en la tabla *Plista* que para ese ítem el campo url coincide.

II. Pruebas sobre el módulo encargado de capturar los ítems de Plista.

a. Los campos asociados a los ítems son capturados correctamente.
b. La inserción de nuevos ítems en la base de datos se realiza satisfactoriamente. Comprobar en la tabla *Plista* que la información coincide con la que se ha insertado.
c. En caso de intentar insertar en la base de datos un ítem ya existente en ella, se actualizan los campos asociados a este. Verificar en la tabla *Plista* volviendo a insertar un ítem ya existente, con algún campo distinto.

5.3 Pruebas de integración en el sistema basado en Twitter

En las siguientes pruebas se verifica que el sistema basado en Twitter funciona adecuadamente y todos sus módulos se conectan correctamente:

I. Las consultas y dominios leídos desde los ficheros de configuración son almacenados con éxito en la base de datos y dicha información se corresponde con la de los ficheros de origen.

II. Las consultas almacenadas en la base de datos son extraídas correctamente. Verificar que los datos extraídos coinciden con el contenido de la tabla *Query*.

III. Añadir un elevado número de consultas (para que se envíen a la API de Twitter una tras otra), verificar que el diseño del sistema impide que se violen las restricciones de tráfico impuestas por la API de Twitter.

IV. Se comprueba que los tweets capturados son almacenados correctamente junto con todas sus variables son insertados correctamente en la base de datos. En el ***Anexo E "Pruebas, búsqueda implementada en el sistema y búsqueda en la web de Twitter"*** se muestra gráficamente el resultado de dicha prueba.

V. Comprobar que el método encargado de actualizar la tabla "Popularity_Recommender" se realiza correctamente. Para ello se probará el sistema con pocos datos y se verificará que, efectivamente, las urls con mayor número de retweets y favoritos. En el ***Anexo C "Generación del ranking de popularidad"*** se detalla dicho método. En el ***Anexo F "Pruebas, estado de la tabla Popularity_Recommender tras su actualización",*** se detalla el resultado de aplicar dicho método sobre los tweets almacenados en la base de datos.

5.4 Pruebas de integración en el sistema basado en Plista

En las siguientes pruebas se verifica que el sistema basado en Plista funciona adecuadamente y todos sus módulos se conectan correctamente:

I. El módulo es capaz de obtener correctamente el dominio asociado a los tweets almacenados a partir del dominio extraído del mensaje enviado por Plista. Para ello se comprobará que los pares de dominios almacenados en la tabla *Code_Translation*, coinciden con los dominios obtenidos.

II. A partir del dominio recibido y los pesos de retweets y favoritos dados, el sistema es capaz de obtener las urls más populares asociadas a dicho dominio. Y a partir de dichas urls buscar los ítems de Plista asociados.

III. El sistema es capaz de enviar a Plista una selección de ítems válida, es decir, verificar que los ítems devueltos corresponden al dominio recibido, para ello utilizar la tabla *Plista* de la base de datos.

IV. En caso de no haber suficientes ítems el sistema utiliza el recomendador de apoyo **Recent Recommender**. Probar que funciona correctamente utilizando una base de datos vacía y verificando que se responde a las peticiones de recomendación.

5.5 Pruebas de sistema y aceptación

En estas pruebas se comprobará que el proyecto en su conjunto funciona correctamente y cumple con los propósitos para los que fue diseñado. Además se verificará el grado de aceptación del producto, es decir, si las recomendaciones ofrecidas por el sistema son interesantes para el usuario. Se han desarrollado en tres entornos de prueba distintos en cuanto a complejidad.

- **Prueba individual** (entorno básico). Consistirá en ofrecer al sistema peticiones de recomendación para un dominio determinado y verificar el correcto funcionamiento del conjunto de ambos sistemas.
- **Prueba con un grupo pequeño de usuarios** (entorno pequeño). Esta prueba consistirá en generar una serie de noticias para un dominio dado y comprobar la opinión de un grupo de usuarios a través de una encuesta.
- **Prueba en un entorno real** (entorno grande). Este entorno real será la plataforma Plista, la cual enviará tanto los ítems recomendables como las solicitudes de recomendación y será la encargada de capturar la respuesta del usuario.

Estas pruebas se han ejecutado de manera satisfactoria. El análisis del grado de aceptación del producto en cada una de ellas se mostrará con detalle en la ***sección 6 "Resultados",*** donde se analizarán los datos obtenidos de las pruebas realizadas en el entorno pequeño y en el entorno real.

6. Resultados

En este apartado se detallarán las condiciones iniciales y los resultados de las pruebas propuestas en la ***sección 5.5 "Pruebas de sistema y aceptación"*** en dos contextos distintos: "**Prueba con un grupo pequeño de usuarios**" y "**Prueba en un entorno real**".

6.1 Usando un grupo pequeño de usuarios

Las pruebas en este entorno consistirán en seleccionar primero un medio de comunicación con presencia en internet y en Twitter y utilizar el recomendador desarrollado en el proyecto para encontrar sus noticias más populares en Twitter. Para ello, en el primer paso, el medio escogido ha sido el **diario** "***20minutos***", y después se han definido las siguientes consultas:

- "**http://www.20minutos.es/**"
- "**20minutos.es**"
- "**20minutos**"

Tras permitir al recomendador recolectar información durante 12 horas, estas han sido las 5 urls asociadas a noticias de "***20minutos***" con mayor popularidad en Twitter:

1. "http://www.20minutos.es/noticia/1346662/0/marido/soraya-saenz-de-santamaria/telefonica/"
2. "http://www.20minutos.es/noticia/2145622/0/bankia/rato/embargo/
3. "http://www.20minutos.es/noticia/2184521/0/alcaldes/sobresueldos/caso-mercurio/"
4. "http://www.20minutos.es/noticia/2183255/0/juanjo-puigcorbe/boadilla-premio-cortos/veto-presiones-politicas/"
5. "http://www.20minutos.es/noticia/2184605/0/pablo-iglesias/libro-entrevista/regular-medios-comunicacion/"

En la siguiente tabla se muestran cada una de las noticias asociadas a las urls anteriores junto con el número de retweets y favoritos correspondientes a cada una de ellas.

Noticia	Retweets	Favoritos
[1] **Telefónica ficha al marido de Soraya Sáenz de Santamaría y a la esposa de Eduardo Madina**	**324**	**57**
[2] **Anticorrupción se opone al embargo de 24.000 millones de euros a Rodrigo Rato y su equipo**	**187**	**31**
[3] **Investigan a 44 alcaldes de toda Cataluña y casi todos los partidos por cobrar sobresueldos**	**68**	**19**
[4] **Culpan al alcalde de Boadilla de vetar a Puigcorbé y Tito Valverde sopesa devolver el premio**	**23**	**12**
[5] **Pablo Iglesias, a favor de regular los medios de comunicación**	**25**	**6**

Tabla 6. Pruebas en entorno pequeño, noticias recomendadas.

El tamaño de la muestra es de ocho usuarios, cinco hombres y tres mujeres con edades comprendidas entre los 18 y los 50 años. La batería de preguntas escogida es la siguiente:

- **Pregunta 1.** ¿Con que frecuencia accedes a contenidos informativos de actualidad? Tanto prensa escrita como Internet, Radio o Televisión.
- **Pregunta 2.** ¿Cuál es la fuente que utilizas habitualmente como canal de comunicación?
- **Pregunta 3.** ¿Te han resultado interesantes las noticias mostradas?
- **Pregunta 4.** ¿Por qué crees que esas noticias te han resultado interesantes?
- **Pregunta 5.** Sobre las noticias que te han resultado interesantes, ¿Crees que hubieras accedido a ellas simplemente viendo su entradilla?
- **Pregunta 6.** ¿Por qué crees que esas noticias no te han resultado interesantes?
- **Pregunta 7.** Observando las noticias disponibles en la web del Publicador, ¿Te hubiera resultado más interesante otra selección de noticias distinta a la recomendada?
- **Pregunta 8.** En cuanto a las noticias recomendadas, ¿Cuál crees que es el elemento que más condiciona para visitar alguna de esas noticias?

En el ***Anexo G*** ***"Pruebas, entorno con un grupo pequeño de usuarios, resultados de la encuesta"*** se muestran los gráficos asociados a los resultados obtenidos para cada una de las preguntas de de la encuesta. Ahora nos vamos a centrar en las dos preguntas que están directamente relacionadas con la evaluación del recomendador propuesto: la 3 (calidad de las recomendaciones) y la 7 (capacidad de mejora del recomendador).

- **Pregunta 3. ¿Te han resultado interesantes las noticias mostradas?**

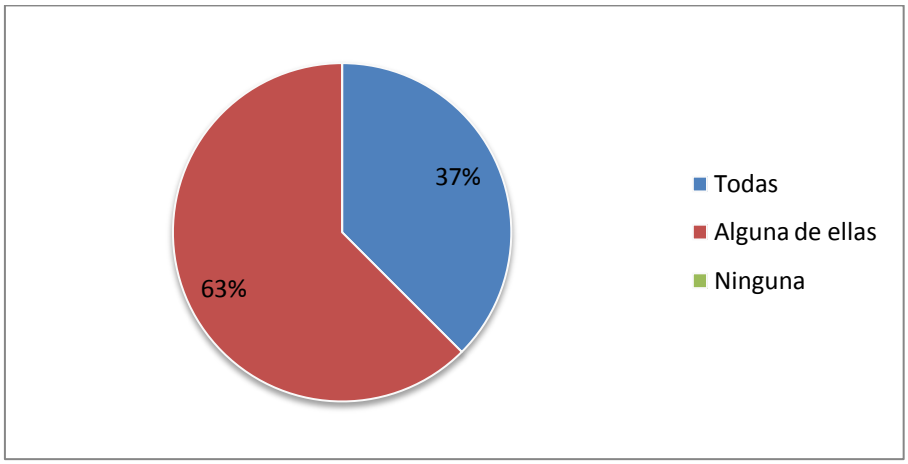

Ilustración 15. Encuesta, pregunta 3, calidad de las recomendaciones.

Como se puede apreciar en la **Ilustración 15**, el usuario muestra interés hacia las noticias recomendadas. En algunos casos incluso a todas ellas.

- **Pregunta 7. Observando las noticias disponibles en la web del publicador, ¿Te hubiera resultado más interesante otra selección de noticias distinta a la recomendada?**

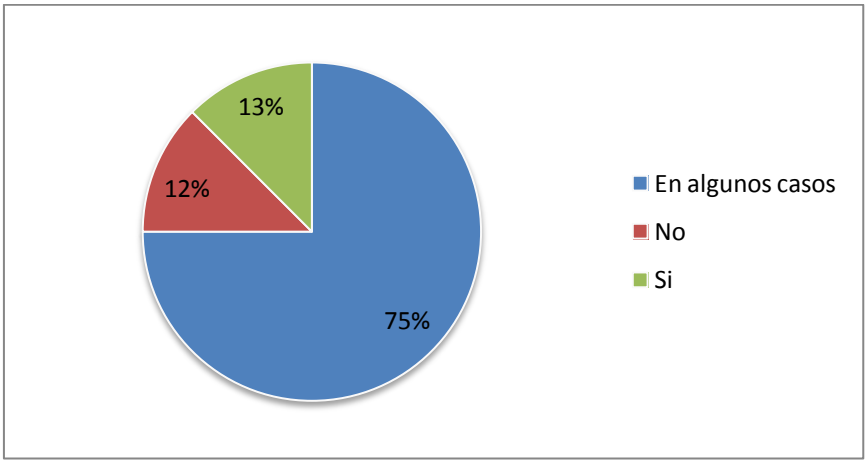

Ilustración 16. Encuesta, pregunta 7, capacidad de mejora del recomendador.

En la **Ilustración 16**, se puede apreciar que la opinión mayoritaria entre los usuarios respecto a las noticias recomendadas, es que en algunos casos les hubieran interesado más otras noticias disponibles en el sitio web del publicador, lo que indica que el recomendador es susceptible de mejora; no obstante, sólo el 13% de los usuarios hubieran preferido un conjunto totalmente diferente de noticias.

En resumen, la evaluación con un grupo pequeño de usuarios nos ha enseñado lo siguiente:

- Internet es una de las principales fuentes de información.
- De la selección de noticias recomendadas siempre hay alguna del interés del usuario, principalmente por su temática.
- Los motivos por los que un usuario no muestra interés en una noticia son variados: su temática, la falta de atractivo de la misma, etc.
- Pese al interés del usuario por una noticia, en muchos casos no hubieran accedido a todas las noticias que les interesaban simplemente viendo la entradilla.

6.2 Entorno real (Plista)

Durante las pruebas en Plista se han atendido cerca de **250.000 peticiones** de recomendación en menos de 10 días, obteniendo cerca de **3.200 clics**, esto supone una tasa cercana al **1.3% de acierto**. En la siguiente tabla se muestra la evolución diaria

	Peticiones	Clics	Tasa (%)
Día 1	**2.145**	**29**	**1,35**
Día 2	**35.975**	**514**	**1,43**
Día 3	**58.574**	**807**	**1,38**
Día 4	**27.465**	**314**	**1,14**
Día 5	**1.877**	**23**	**1,23**
Día 6	**8.239**	**121**	**1,47**
Día 7	**52.868**	**540**	**0,95**
Día 8	**13.146**	**103**	**0,78**
Día 9	**45.663**	**703**	**1,54**
TOTAL	**245.952**	**3.154**	**1,28**

Tabla 7. Evolución diaria del recomendador en Plista.

Estos resultados se pueden ver gráficamente en la siguiente figura que incluye tanto el **número de peticiones diarias** como el **número de clics diarios** y la **tasa media de acierto** diaria (**CTR**, de Click Through Rate, en inglés):

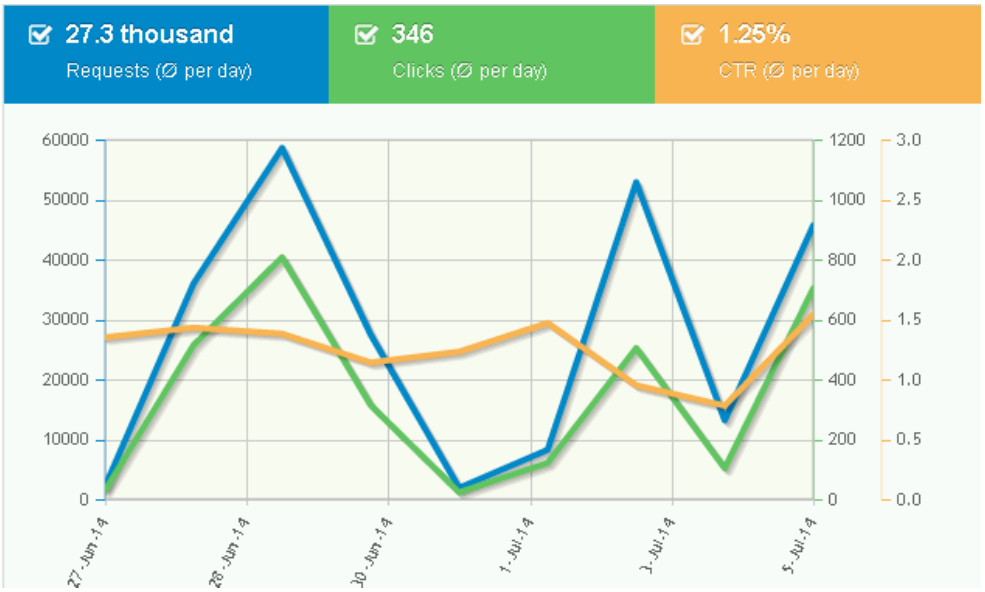

Ilustración 17. Evolución diaria del recomendador en Plista.

Para averiguar cómo de significativo es el impacto de estos resultados en el entorno de Plista, registramos en el sistema un recomendador base durante el mismo período. El recomendador elegido fue el Recent Recommender. Su evolución se puede observar en la Ilustración 18, donde se puede comparar la diferente tasa de acierto entre ellos (mejor para el Recent Recommender), y, de manera más significativa, el mayor número de peticiones de recomendación. Esto se puede deber a cómo funciona internamente Plista y que provoca que exista un comportamiento del tipo "*rich gets richer*": cuando Plista tiene que elegir a cuál de los recomendadores registrados mandar una petición, tiene en cuenta sus aciertos anteriores, por lo que cuanto mejor lo hace, más peticiones recibe.

Hay que mencionar que el 1 de Julio hubo un problema en el servidor que alojaba los dos recomendadores, lo cual explica la caída en ambos. Salvo ese día, se puede observar que los dos métodos son bastante estables, aunque el Recent Recommender es ligeramente mejor que el propuesto. Esto es probable que se deba a la falta de cobertura del Twitter Recommender, hemos observado que no existen muchos resultados para las consultas presentadas en el Anexo B, por lo que nuestro recomendador tiende a mostrar pocas sugerencias, siendo en su mayoría las mismas.

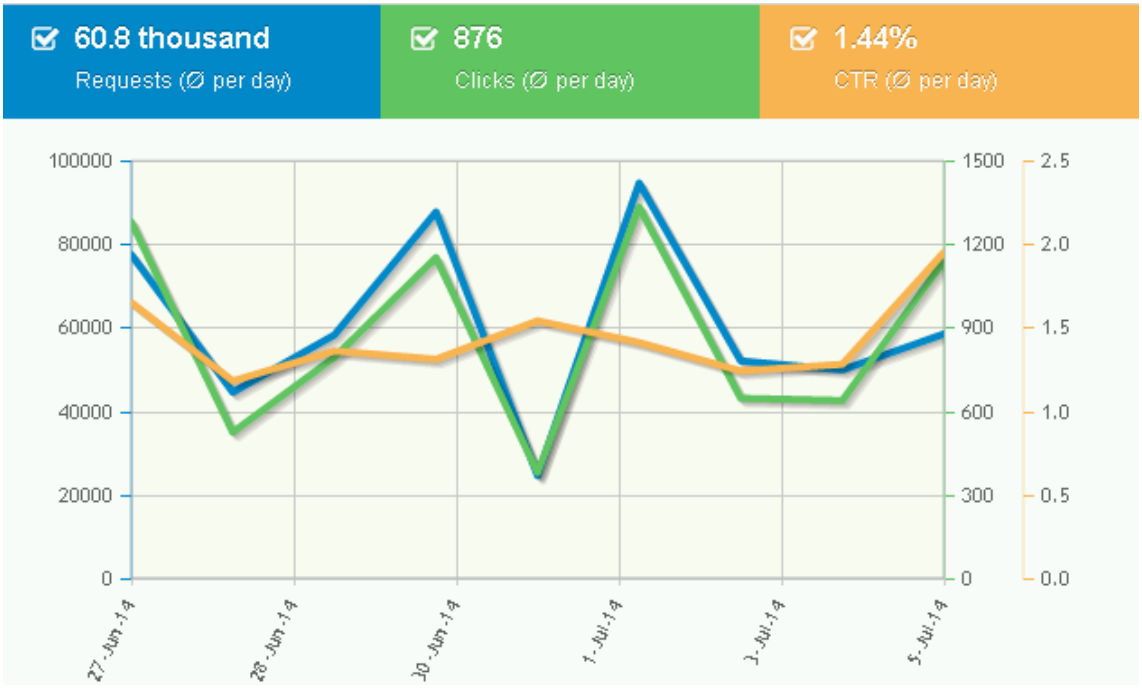

Ilustración 18. Evolución diaria del Recent Recommender en Plista.

También hicimos pruebas con los parámetros del Twitter Recommender (es decir, con el peso que reciben los retweets y los favoritos al calcular la popularidad). Para ello, registramos dos recomendadores más durante un día, uno que sólo utilizaba la información de los retweets y otro sólo la información de los favoritos. Al cabo de ese día, ambos recibieron una cantidad comparable de peticiones de recomendación así como un número similar de clics:

- El recomendador que sólo usa retweets recibió 5.974 peticiones y se obtuvieron 64 clics, es decir, una tasa de 1,07% de acierto.

- El recomendador que sólo usa favoritos recibió 6.248 peticiones y se obtuvieron 66 clics, es decir, una tasa de 1,06% de acierto.

Esta prueba nos mostró que usando sólo uno de los campos se consiguen peores resultados que al usar los dos (comparar con la **Tabla 7**) y que ninguno de ellos es significativamente mejor que el otro.

7. Conclusiones y Trabajo Futuro

En este Trabajo de Fin de Grado hemos considerado la recomendación de noticias usando una red social como Twitter. Los recomendadores de noticias son cada vez más frecuentes en la actualidad, sin embargo, el uso de Twitter, concretamente de las urls contenidas en los tweets, como fuente para medir la popularidad de una determinada noticia en un momento dado, es relativamente novedoso.

Una restricción que hemos tenido en cuenta desde el principio – y abordado en consecuencia – en el proyecto ha sido el entorno real en el que se iba a probar finalmente el recomendador. Este entorno venía definido por Plista, una plataforma que permite a desarrolladores realizar sus recomendaciones a usuarios reales a través de una API, cuyo funcionamiento se basa en un sencillo concepto: un usuario accede a la dirección web de un publicador asociado a Plista, este publicador requiere la recomendación de noticias, dicha petición es enviada a Plista junto con cierta información acerca del usuario, Plista a su vez deriva la petición a alguno de los recomendadores registrados en su plataforma, el cual devuelve sus recomendaciones, las cuales siguen el mismo camino de vuelta al usuario.

Basándose en estos conceptos se ha desarrollado un proyecto cuyo objetivo final es responder a las peticiones de recomendación recibidas desde Plista devolviendo los ítems más populares en Twitter en ese momento. El proyecto está compuesto por varios módulos, los cuales, pueden agruparse en dos grandes conjuntos, por un lado un conjunto centrado en la interacción con Twitter y por otro lado un conjunto basado en la interacción con Plista. Los módulos que componen el proyecto son los siguientes:

- **Extracción de datos desde Twitter**. Este componente es el encargado de poblar la base de datos con tweets actuales, para lo cual, a través de la librería Twitter4J, la aplicación interactúa con la API de Twitter, y usando SQLite, se almacenan los tweets recibidos en una base de datos local.
- **Tratamiento de información y generación de rankings.** A partir de la información extraída previamente de Twitter, este componente es el encargado del tratamiento de las urls contenidas en cada tweet y la elaboración del ranking de popularidad a partir de los retweets y favoritos asociados a cada url.

- **Extracción de datos desde Plista**. Este componente recibe los "***Item Updates***" a través de la API de Plista, a través de estos "updates" se pueden capturar nuevos "ítems" (noticias) de Plista o actualizaciones de los ya existentes. Estos cambios son insertados en la base de datos.
- **Tratamiento de peticiones de recomendación.** Con gran frecuencia la API de Plista envía peticiones a nuestro recomendador, por lo que este componente se encarga de estar a la espera de recibir dichas peticiones. Tras recibir una petición de recomendación, busca en la base de datos las recomendaciones más populares en ese momento para el dominio del publicador que solicitó dicha recomendación.

El proyecto ha resultado un interesante punto a tener en cuenta a la hora de desarrollar futuros recomendadores. Como conclusión del análisis de los resultados obtenidos cabe resaltar la importancia de que el publicador de noticias tenga una importante presencia en Twitter, para que las recomendaciones basadas en la popularidad de esta red social sean de calidad.

En cuanto a la parte formativa, en el desarrollo de este proyecto se han empleado los conocimientos desarrollados durante la carrera, han sido especialmente importantes los conocimientos adquiridos sobre programación en Java y manejo de bases de datos SQL, además se han adquirido diferentes conocimientos acerca de nuevas plataformas y tecnologías, entre estos nuevos conocimientos se destacan:

- **Sistemas de recomendación.**
- **API Twitter,** donde se ha utilizado la librería **Twitter4j** para interactuar con dicha API desde el proyecto Java.
- **API Plista,** donde se ha empleado el **Proyecto Recommenders** como punto de partida para tratar con dicha API.
- **JSON,** para la captura y tratamiento de los datos recibidos desde Plista.
- **SQLite,** en la gestión de la base de datos utilizada en el proyecto.
- **Maven,** utilizado en la gestión del proyecto Java, incluyendo sus dependencias de otros proyectos y librerías.

Respecto al trabajo futuro sobre este proyecto, podrían aplicarse diversas mejoras con el objetivo de incrementar su aceptación por parte de los usuarios, es decir, el interés que estos muestran hacia las sugerencias de recomendación presentadas. Algunas de las mejoras más relevantes serían:

- **Recomendaciones personalizadas**, es decir, recomendaciones basadas en los perfiles de cada usuario: grupo de edad, género, etc.
- **Recomendaciones temáticas**, es decir, a la hora de sugerir una recomendación al usuario dar importancia al ítem que está viendo en ese momento.
- **Recomendaciones combinadas,** observar la efectividad del desarrollado junto con otros recomendadores existentes en el ***proyecto plistarecs,*** por ejemplo el ***Lucene Recommender*** o el ***Category-based Recommender***.
- **Recomendador de última hora,** es decir, recomendar noticias que sean tendencia, por ejemplo en la última hora. Para ello descartar todos los tweets cuya antigüedad sea superior a ese periodo de tiempo establecido.
- **Otros dominios,** sería aconsejable comprobar cuánto dependen los resultados obtenidos de los dominios a recomendar, en particular, de su mayor o menor presencia en Twitter.

Finalmente, como resultado de este Trabajo de Fin de Grado, se ha determinado que el código de la implementación estará disponible en el siguiente repositorio de GitHub:

https://github.com/adrianotero/TFG_TwitterRecommender

Usuario: adrianotero; repositorio: TFG_TwitterRecommender

8. Referencias

En esta sección se detalla la principal bibliografía consultada durante la elaboración de este trabajo de fin de grado.

[1] *API Twitter, Documentation* (n.d.) Visitado Febrero de 2014 desde: https://dev.twitter.com/docs

[2] *Categorical Index Of SQLite Documents* (n.d.) Visitado Marzo de 2014 desde: http://www.sqlite.org/docs.html

[3] *JavaScript Object Notation (JSON)* (n.d.) Visitado Abril de 2014 desde: http://json.org/

[4] *Maven, Frequently Asked Technical Questions* (2014) Visitado Febrero de 2014 desde: http://maven.apache.org/general.html

[5] *Maven: using Netbeans with Apache Maven* (2013) Visitado Febrero de 2014 desde: http://wiki.netbeans.org/Maven

[6] *Netbeans, General Java Development Learning Trail* (n.d.) Visitado Febrero de 2014 desde: https://netbeans.org/kb/trails/java-se.html

[7] *Recommenders/plistaclient, GitHub* [Software]. Disponible desde: https://github.com/recommenders/plistaclient

[8] *Recommenders/plistarecs, GitHub* [Software]. Disponible desde: https://github.com/recommenders/plistarecs

[9] *REST API v1.1 Resources* (n.d.) Visitado Febrero de 2014 desde: https://dev.twitter.com/docs/api/1.1

[10] *SQLite Library, Frequently Asked Questions* (n.d.) Visitado Marzo de 2014 desde: http://www.sqlite.org/faq.html

[11] *The Twitter Streaming API* (2012) Visitado Febrero de 2014 desde: https://dev.twitter.com/docs/api/streaming

[12] *Twitter4J Library, Frequently Asked Questions* (n.d.) Visitado Febrero de 2014 desde: http://twitter4j.org/en/faq.html

[13] *Twitter4J Library, Javadoc* (n.d.) Visitado Febrero de 2014 desde: http://twitter4j.org/javadoc/twitter4j/Twitter.html

[14] *Twitter Help Center* (2014) Visitado Febrero de 2014 desde: https://support.twitter.com/

[15] *Using the Twitter Search API* (2013) Visitado Febrero de 2014 desde: https://dev.twitter.com/docs/using-search

[16] T. Brodt. *Open Recommendation Platform (ORP)* [Presentación de Google Docs] (2014) Visitado Mayo de 2014 desde: https://docs.google.com/presentation/d/1UuqLS5WPmgUrWyWn4qHP0gvFbvO-Dg2LeZj60tG7LYs/present?pli=1&ueb=true#slide=id.gf588b212_00

[17] T. Brodt, T. Heintz, A. Bucko, A. Palamarchuk. *ORP Protocol* (2013) Visitado Mayo de 2014 desde http://orp.plista.com/documentation/download

[18] J. Chen, R. Nairn, L. Nelson, M. Bernstein, E.H. Chi. *Short and Tweet: Experiments on Recommending Content from Information Streams* (2010) In 28th International Conference on Human Factors in Computing Systems (CHI 2010). Atlanta, US. Disponible desde: http://hci.stanford.edu/publications/2010/zerozero88/zerozero88-chi2010.pdf

[19] A. Said, A. Bellogín, A. de Vries. *News Recommendation in the Wild: CWI's Recommendation Algorithms in the NRS Challenge* (2013) In International News Recommender Systems Challenge (NRS 2013), at the 7th ACM Conference on Recommender Systems (RecSys 2013). Hong Kong, China. Disponible desde: http://ir.ii.uam.es/~alejandro/2013/nrs.pdf

[20] P. Vanek. *The Sqliteman Handbook (2007)* Visitado Marzo de 2014 desde: http://docs.sqliteman.com/user/en/

Anexos Técnicos

A. Proceso de autentificación en la API de Twitter

Twitter utiliza OAuth para dar a las aplicaciones externas permisos de acceso a su API, esto permitirá a dichas aplicaciones interactuar con la API sin comprometer los datos de los usuarios.

En este anexo se detalla el proceso por el cual una aplicación es registrada en la API de Twitter con el objetivo de integrar el proceso de autentificación en nuestra aplicación a través del protocolo OAuth y con la ayuda de la librería Twitter4J. La operación de integrar el proceso de autentificación en la aplicación es necesaria puesto que es un requisito para realizar la búsqueda de tweets mediante la Search API.

El primer paso consiste en registrarse en Twitter, para ello se debe acceder a la dirección https://twitter.com/ y crear una cuenta de Twitter. Tras esto es necesario acceder a la web de desarrolladores de Twitter https://dev.twitter.com/ y acceder a la cuenta creada previamente en Twitter.

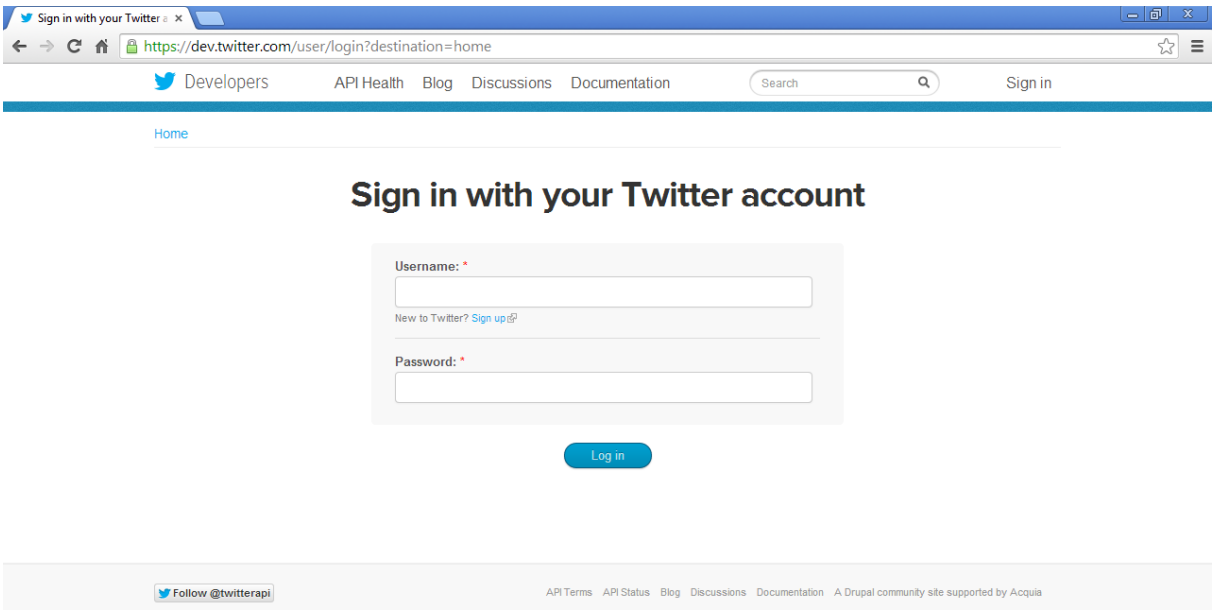

Ilustración 19. Desarrolladores de Twitter, acceso web.

El siguiente paso consistirá en acceder a las aplicaciones asociadas a la cuenta, se debe acceder a través de ***"My applications"***.

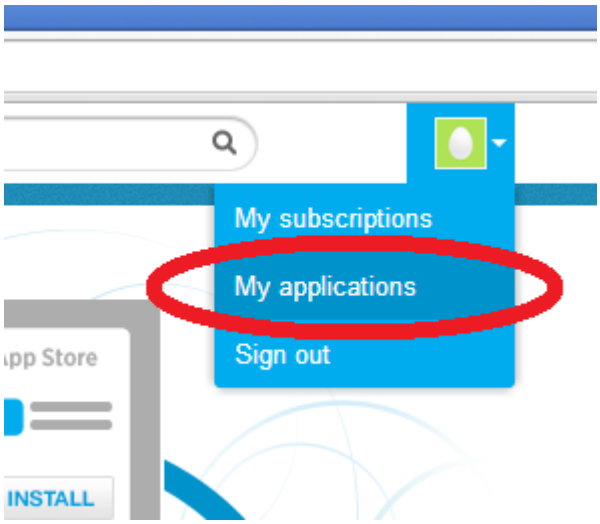

Ilustración 20. Desarrolladores de Twitter, aplicaciones.

Tras acceder a nuestras aplicaciones es necesario crear una nueva aplicación, podemos acceder a este menú a través del botón ***"Create New App"***.

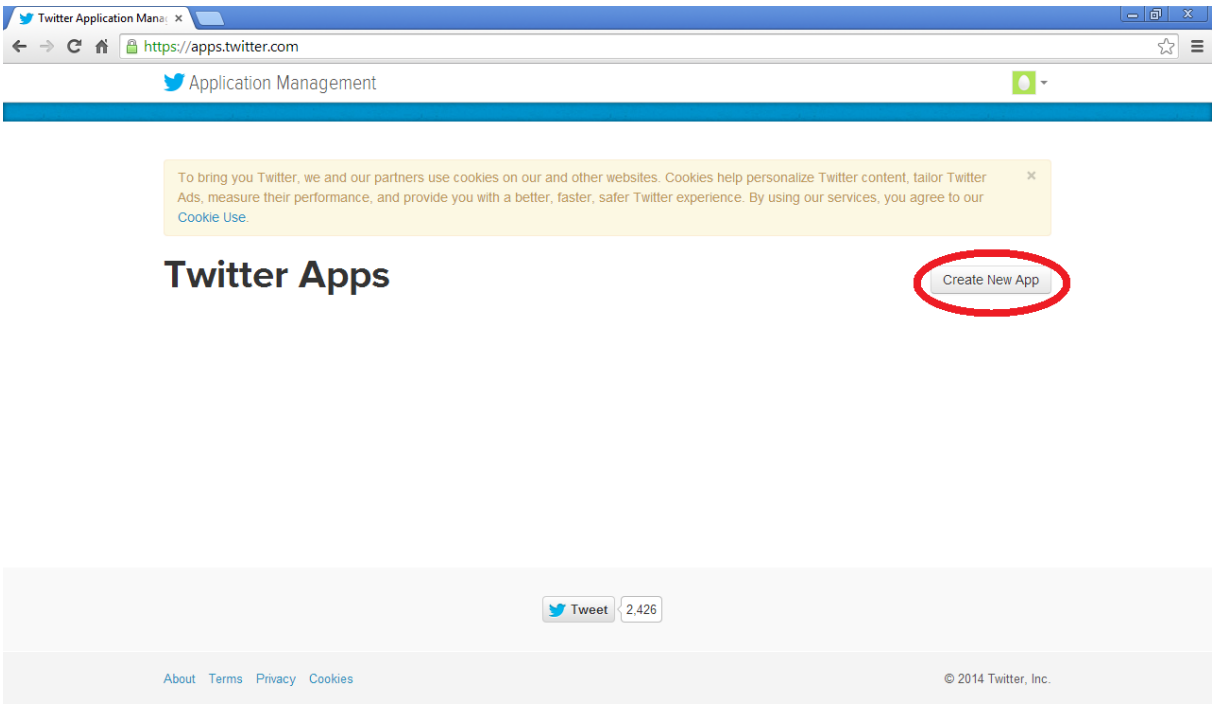

Ilustración 21. Desarrolladores de Twitter, nueva aplicación.

A continuación se deben introducir los datos de la aplicación, de estos son indispensables: el nombre de la aplicación, su descripción y una dirección web asociada. También será indispensable aceptar las condiciones requeridas por el servicio.

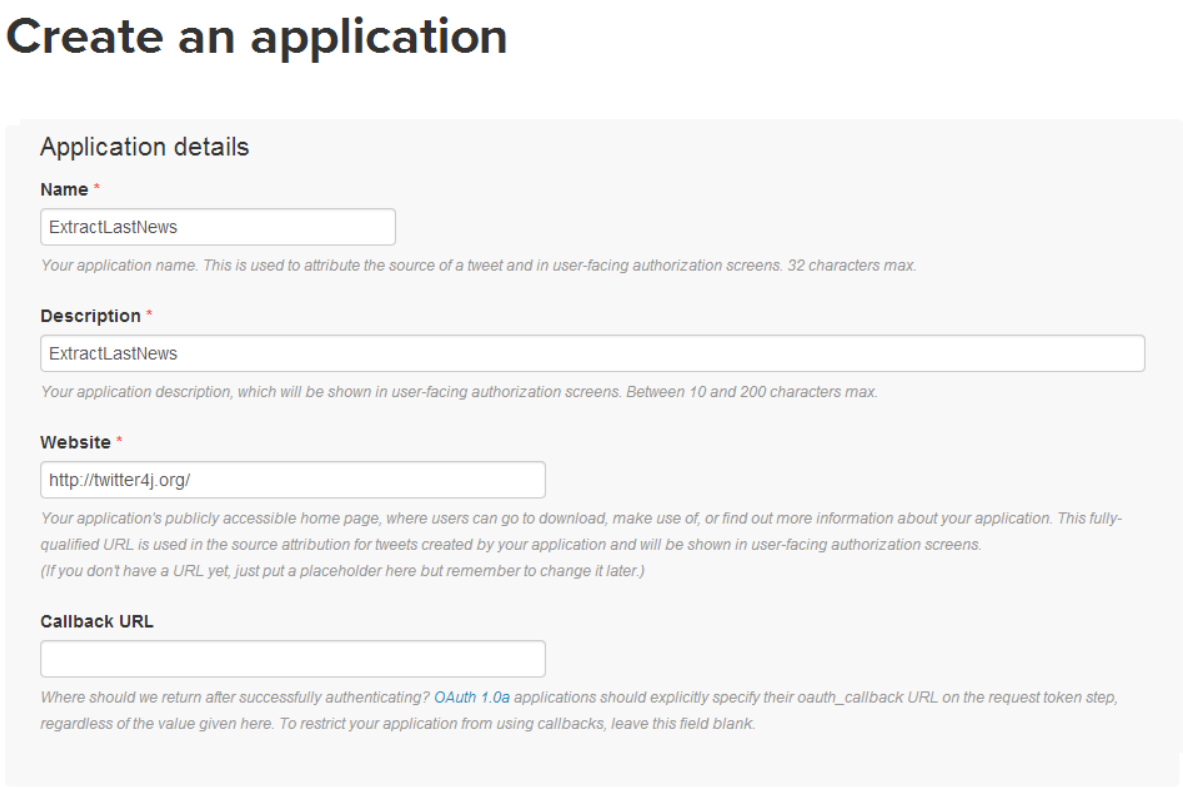

Ilustración 22. Desarrolladores de Twitter, crear aplicación.

Una vez registrada la aplicación se tendrán acceso a sus detalles, lo cuales podrán ser modificados en cualquier momento.

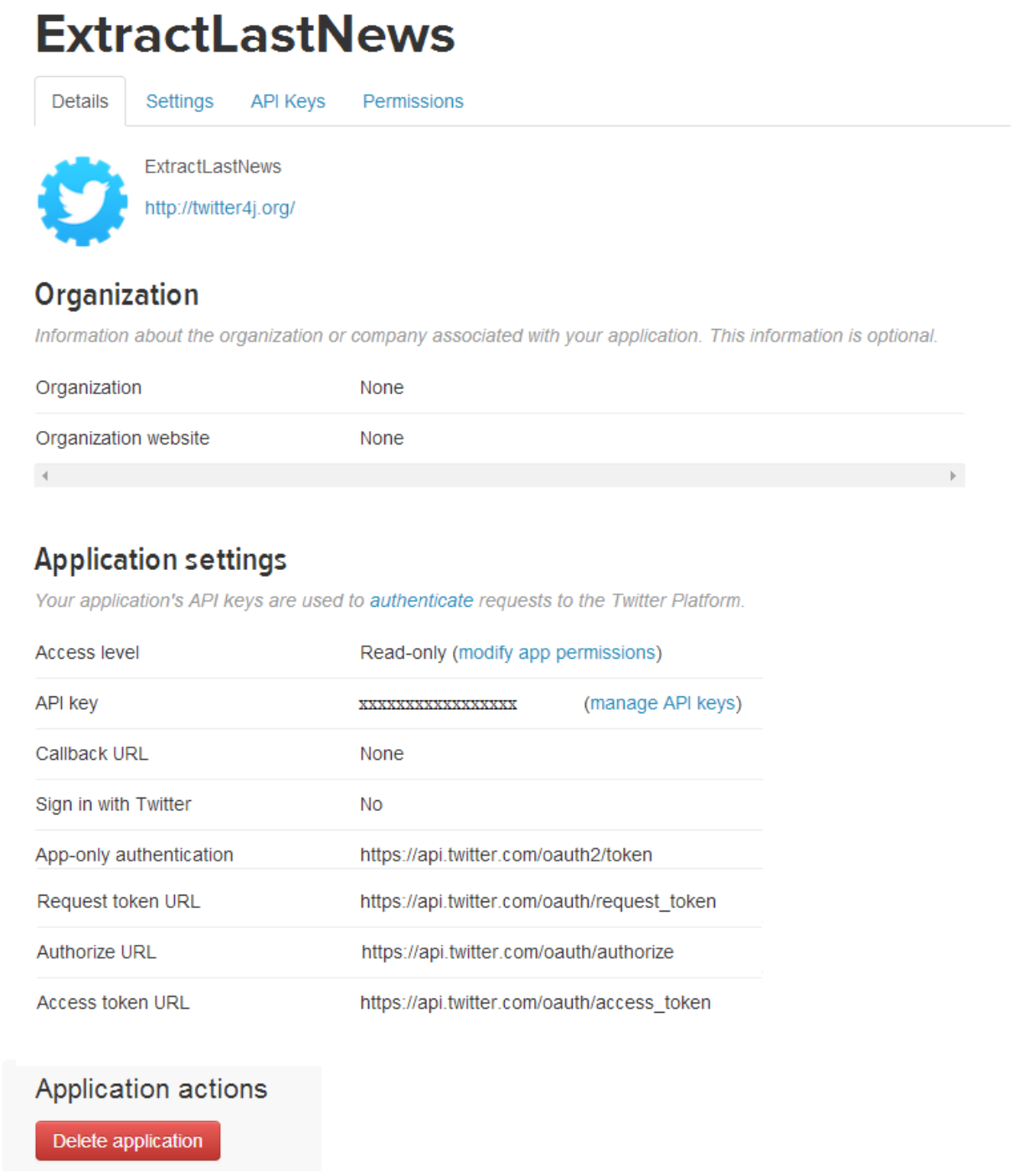

Ilustración 23. Desarrolladores de Twitter, detalles de la aplicación.

Entre estos detalles se encuentran las claves de acceso necesarias para acceder utilizando la librería Twitter4j a la API de Twitter. Esta información se puede encontrar en la pestaña ***"API Keys"***.

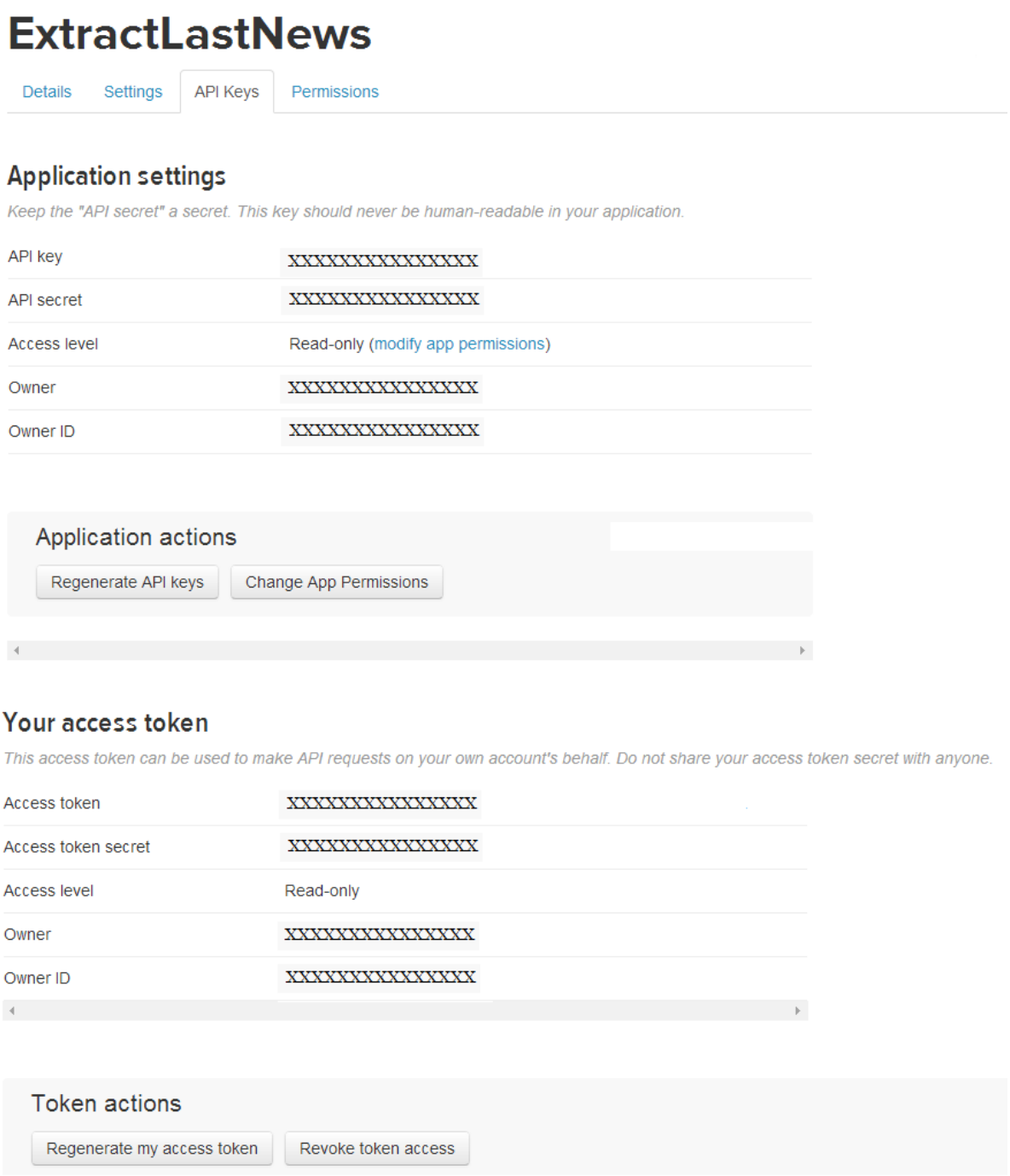

Ilustración 24. Desarrolladores de Twitter, claves de acceso (API Twitter).

Finalmente es necesario incluir estas claves en nuestra aplicación al comienzo de la clase encargada de extraer la información desde Twitter a través de la librería Twitter4j para así poder utilizar la API de Twitter.

```
//Twitter Conf.
System.setProperty("twitter4j.loggerFactory", "twitter4j.NullLoggerFactory");
twitter4j.conf.ConfigurationBuilder cb = new twitter4j.conf.ConfigurationBuilder();
cb.setDebugEnabled(DEBUG);
cb.setDebugEnabled(DEBUG).setOAuthConsumerKey("XXXXXXXXXXXXXXXXXXXXX");
cb.setDebugEnabled(DEBUG).setOAuthConsumerSecret("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
cb.setDebugEnabled(DEBUG).setOAuthAccessToken("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
cb.setDebugEnabled(DEBUG).setOAuthAccessTokenSecret("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");


TwitterFactory tf = new TwitterFactory(cb.build());
twitter = tf.getInstance();
```

Ilustración 25. Autenticación en la API de Twitter a través de la librería Twitter4J

B. Ficheros de carga inicial de la Base de Datos

A continuación se muestra la información básica que deberá ser cargada en la base de datos asociada al proyecto, esta información deberá modificarse si posteriormente Plista actualiza sus publicadores asociados.

En esta tabla se incluyen cada uno de los dominios sobre los que se buscará información en Twitter. Se incluye tanto el identificador de dicho dominio como su nombre asociado.

Identificador del Dominio	Nombre del Dominio
418	Ksta
596	Sport1
694	Gulli
1677	Tagesspiegel
2522	Computerwoche
2524	Cio
2525	Cfoworld
3336	Tecchannel
12935	Wohen-und-garden
13554	Motor-talk

Tabla 8. Carga Inicial, dominios de publicadores.

En la siguiente tabla se incluirán cada una de las consultas que se realizarán contra la API de Twitter y que devolverán los tweets asociados a alguno de los publicadores de noticias.

Identificador de la Consulta	Texto de la consulta	Identificar del dominio asociado a dicha consulta
10	http://www.ksta.de	**418**
11	ksta.de	**418**

20	http://www.sport1.de	**596**
21	sport1.de	**596**
30	www.gulli.com	**694**
40	www.tagesspiegel.de	**1677**
50	www.computerwoche.de	**2522**
60	www.cio.de	**2524**
70	www.cfoworld.de	**2525**
80	www.tecchannel.de	**3336**
90	www.wohnen-und-garten.de	**12935**
100	www.motor-talk.de	**13554**

Tabla 9. Carga inicial, consultas de Twitter.

En la siguiente tabla se incluye la forma en que se asocian cada uno de los identificadores de los dominios en Twitter con sus correspondientes en Plista. Como se puede observar, esta asociación corresponde a la identidad, pero permite generalidad en el caso en que se quiera utilizar dominios externos u otra funcionalidad en el futuro.

Identificador asociado al dominio en Twitter	Identificador asociado al dominio en Plista	Categoría del dominio
418	**418**	News
596	**596**	Hobby
694	**694**	Technology
1677	**1677**	News
2522	**2522**	Technology
2524	**2524**	Business
2525	**2525**	Business
3336	**3336**	Technology
12935	**12935**	Hobby
13554	**13554**	Hobby

Tabla 10. Carga inicial, correspondencia Twitter-Plista.

C. Generación del ranking de popularidad (implementación)

En la siguiente figura se muestra el código mediante el que se implementa la actualización del ranking de noticias más populares:

```
public void RefreshPopularityRec(){
    String query = null;

    try {
        connection.setAutoCommit(false);

        Statement statement = connection.createStatement();
        statement.setQueryTimeout(30);  // set timeout to 30 sec.

        query = "DROP TABLE IF EXISTS TEMP;";
        statement.executeUpdate(query);
        query = "DROP TABLE IF EXISTS TEMP2;";
        statement.executeUpdate(query);

        query = "CREATE TABLE IF NOT EXISTS TEMP " +
                "(URL           CHAR(140)    NOT NULL, " +
                " DOMAINID      LONG         NOT NULL, " +
                " RETWEETS      INT          NOT NULL, " +
                " FAVOURITES    INT          NOT NULL)";
        statement.executeUpdate(query);

        /* Captura todos los tweets almacenados en la base de datos */
        List<TweetData> TweetList_read = this.Select_Tweets("");
        for (TweetData tweet : TweetList_read) {

            String[] urls = tweet.getUrls().split("\n");
            for (String url : urls) {

                if (!url.equals("")) {
                    // Tratar por separado cada url del tweet
                    query = ("INSERT INTO TEMP VALUES ("
                            + "'" + url + "', "
                            + tweet.getDomainid() + ", "
                            + tweet.getRetweets() + ", "
                            + tweet.getFavoritos() + ");");

                    statement.executeUpdate(query);
                }
            }
        }
```

Ilustración 26. Implementación del proceso de generación del ranking de popularidad.

```
        // Agrupar en función de la URL
        query =   "CREATE TABLE IF NOT EXISTS TEMP2 AS "
                + "SELECT URL, DOMAINID, SUM(RETWEETS) AS TOT_RETWEETS, "
                + "SUM(FAVOURITES) AS TOT_FAVOURITES "
                + "FROM TEMP "
                + "GROUP BY URL, DOMAINID "
                + "ORDER BY (TOT_RETWEETS + TOT_FAVOURITES) DESC;";

        statement.executeUpdate(query);

        // Insertar los nuevos valores calculados en la BD
        this.Clean("POPULARITY_RECOMMENDER");
        query = "INSERT INTO POPULARITY_RECOMMENDER SELECT URL, DOMAINID, "
                + "TOT_RETWEETS, TOT_FAVOURITES FROM TEMP2;";
        statement.executeUpdate(query);

        statement.close();
        connection.commit();

    } catch (Exception e) {
        System.err.println(query);
        System.err.println(e.getClass().getName() + ": " + e.getMessage());
    }
}
```

Ilustración 27. Implementación del proceso de generación del ranking de popularidad (continuación).

D. Clase AbstractCombinationRecommender, método recommend (implementación)

Mediante el método ***recommend*** de la clase AbstractCombinationRecommender el sistema implementa el proceso encargado de generar las peticiones de recomendación utilizando el **Twitter Recommender** como **base** y el **Recent Recommender** como **apoyo** en caso de que no se hayan obtenido suficientes sugerencias de recomendación. A continuación se detalla el código mediante el que se implementa dicho método:

```
public List<Long> recommend(Message input, Integer limit) {
    if (recommenders.length > 2) {
        throw new IllegalArgumentException("Too many recommenders ('recommend' method needs to be overridden): "
                                           + recommenders.length);
    }
    Recommender recommender = recommenders[0];
    Recommender backupRecommender = null;
    if (recommenders.length == 2) {
        backupRecommender = recommenders[1];
    }
    final List<Long> recList = recommender.recommend(input, limit);
    // we complete the recommendation list if a backup recommender was provided
    if (backupRecommender != null && recList.size() < limit) {
        List<Long> backupList = backupRecommender.recommend(input, BACKUP_LENGTH * limit);
        for (Long item : backupList) {
            if (recList.size() < limit && !recList.contains(item)) {
                recList.add(item);
            }
        }

    }
    return recList;
}
```

Ilustración 28. Clase AbstractCombinationRecommender, método recommend.

Fuente:

{https://github.com/recommenders/plistarecs/blob/master/src/main/java/net/recommenders/plista/rec/combination/AbstractCombinationRecommender.java"}

E. Clase TwitterRecommender, métodos de recomendación de ítems y actualización de la tabla hash (implementación)

La siguiente figura muestra la implementación del método encargado de gestionar los ítems que serán recomendados por el sistema de recomendación de noticias basado en la popularidad de dichos ítems en Twitter, este proceso se detalla en la ***sección 4.2.4.1 "Gestión de las peticiones de recomendación en Plista"***.

```
@Override
public List<Long> recommend(Message _input, Integer limit) {
    tempLog.println("recommend_in\t" + _input);

    final Long domain = _input.getDomainID();
    if (domain == null) {
        return new ArrayList<Long>(0);
    }
    List<Long> recommendations = cacheDomainRecs.get(domain);

    if ((recommendations == null) || (recommendationRequests >= MAX_REC_REQ_TO_CACHE)) {
        final int maxLimit = 10 * limit;
        new Thread() {

            @Override
            public void run() {
                List<Long> recommendations = new ArrayList<Long>();
                getRecommendationsUsingDB(domain, recommendations, maxLimit);
                cacheDomainRecs.put(domain, recommendations);
                recommendationRequests = 0;
            }
        }.start();
    } else {
        recommendationRequests++;
    }
    if (recommendations.size() > limit) {
        recommendations = recommendations.subList(0, limit);
    }
    tempLog.println("recommend_out\t" + recommendations);
    return recommendations;
}
```

Ilustración 29. Clase TwitterRecommender, método recommend.

La siguiente figura muestra el proceso encargado de generar los ítems que son utilizados en el método anterior para actualizar el contenido de la tabla hash. Este método consultará la base de datos para llevar a cabo la actualización.

```
private void getRecommendationsUsingDB(Long domain, List<Long> recommendations, Integer limit) {
    Long true_domain = -11;
    List<String> urls = null;
    int count = 0;
    PLista item = null;

    if (tweetdb != null) {

        // Buscar a partir del domainid dado por PLista el respectivo en el recomendador
        true_domain = tweetdb.translatePListaDomain(domain);

        // Preguntar al recomendador por las urls para ese dominio
        urls = tweetdb.getUrlsOrderedByPopularity(true_domain, favWeight, retWeight);

        for (String url : urls) {
            if (count < limit) {

                // obtener el id correspondiente (tabla plista)
                item = tweetdb.searchPListaURL(url);
                if (item != null) {
                    // y comprobar flag (es recomendable?)
                    if (item.getFlag_Recommendable() == 1) {
                        recommendations.add(item.getItemId());
                        count++;
                    }
                }
            }
        }
    }
}
```

Ilustración 30. Clase TwitterRecommender, método getRecommendationsUsingDB

E. Pruebas, búsqueda implementada en el sistema y búsqueda en la web de Twitter

En las siguientes figuras se muestra el resultado de realizar una consulta, en este ejemplo será "***http://www.20minutos.es/"***, por un lado mediante los métodos implementados en el sistema (a través de la API de Twitter), Ilustración 31, y por otro lado utilizando el buscador facilitado por la web de Twitter (Ilustración 32).

Ilustración 31. Resultado de una consulta utilizando el buscador facilitado por la web de Twitter.

	RETWEETS	FAVOURITES	TEXT	USERNAME
18998	0	0	"@20m: El extesorero del #PP Álvaro #Lapuerta afirma que nunca se ha apropiado de...	Moran_Fernandez
18999	0	1	La Fiscalía Anticorrupción pide imputar al sobrino de Felipe González en el caso...	Egarenca
19000	3	0	El extesorero del #PP Álvaro #Lapuerta afirma que nunca se ha apropiado de dinero... JAJAJA #ascodepeperos	MARIAJ_SALADO
19001	0	0	@20m: El extesorero del #PP Álvaro #Lapuerta afirma que nunca se ha apropiado de...	Antonio_314
19002	9	2	#ÚLTIMAHORA El extesorero del #PP Álvaro #Lapuerta afirma que nunca se ha...	20m
19003	0	0	Sólo 650.000 viviendas han obtenido el certificado energético en su primer año en vigor...	gabrielavera08
19004	0	0	Sólo 650.000 viviendas han obtenido el certificado energético en su primer año en vigor...	josefina_silva9

Ilustración 32. Resultado de una consulta utilizando la implementación de búsquedas en la API de Twitter.

Como se puede apreciar en dichas imágenes ambas búsquedas devuelven los mismos resultados por lo que se puede concluir que la búsqueda implementada utilizando la API de Twitter funciona correctamente.

F. Pruebas, tweet almacenado en el sistema y tweet almacenado en la web de Twitter

En las siguientes figuras se muestra el resultado de comparar un tweet almacenado en la web de Twitter respecto al mismo tweet almacenado en nuestra base de datos. Para poder comparar el mismo tweet se ha extraído un tweet de la base de datos y se ha accedido al almacenado en la web de Twitter a través de la url de dicho tweet, para construir dicha url, la cual no es proporcionada por ningún método de la librería Twitter4J, se ha usado la siguiente fórmula:

- **http://twitter.com/** + **<nombre de usuario>** + **/status/** + **<id del tweet>**

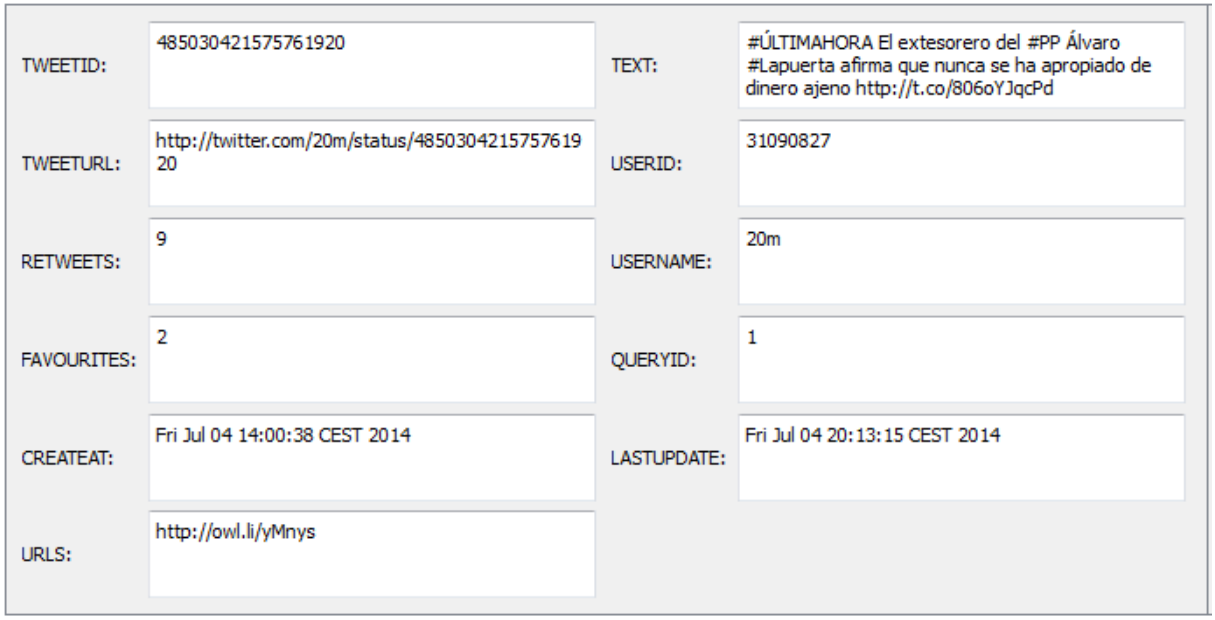

Ilustración 33. Tweet extraído de la base de datos.

En el tweet del ejemplo anterior se puede ver cómo el nombre de usuario (**username**) es "***20m***" y el id del tweet (**tweetid**) es "***485030421575761920***", por lo tanto, como se muestra en la figura anterior, la url de dicho tweet (**tweeturl**) es "***http://twitter.com/20m/status/485030421575761920***". Mediante dicha url se accederá al tweet almacenado en la web de Twitter, como se muestra en la siguiente figura:

Ilustración 34. Tweet accedido a través de la web de Twitter.

En este ejemplo se puede apreciar que todos los campos asociados a dicho tweet se corresponden con los valores almacenados en la base de datos, concretamente en la figura anterior se pueden apreciar los siguientes elementos:

1. **USERNAME**
2. **TEXT**
3. **RETWEETS**
4. **FAVOURITES**
5. **CREATEAT**

F. Pruebas, estado de la tabla Popularity_Recommender tras su actualización

En la siguiente figura se muestra el estado de la tabla **Popularity_Recommender** tras aplicar el método de actualización de la misma, utilizando los tweets almacenados en la tabla **Historical**. Este método de actualización se detalla en el ***Anexo C*** ***"Generación del ranking de popularidad".***

	URL	DOMAINID	RETWEETS	FAVOURITES
1	http://www.20minutos.es/noticia/1346662/0/marido/soraya-saenz-de-santamaria/telefonica/	1	324	57
2	http://www.20minutos.es/noticia/2145622/0/bankia/rato/embargo/#xtor=AD-15&xts=467263	1	187	31
3	http://www.20minutos.es/noticia/2184521/0/alcaldes/sobresueldos/caso-mercurio/	1	38	9
4	http://www.20minutos.es/noticia/2183255/0/juanjo-puigcorbe/boadilla-premio-cortos/veto-presiones-politicas/	1	15	6
5	http://www.20minutos.es/noticia/2184605/0/pablo-iglesias/libro-entrevista/regular-medios-comunicacion/	1	8	9
6	http://www.20minutos.es/noticia/2145622/0/bankia/rato/embargo/	1	10	5
7	http://www.20minutos.es/deportes/noticia/seleccion-argelia-futbol-donacion-primas-gaza-mundial-brasil-2014-2184379/0/	1	10	4

Ilustración 35. Estado de la tabla Popularity_Recommender tras una actualización.

Observando la tabla, se puede observar a primera vista que los tweets más populares, aquellos con mayor número de retweets y favoritos, son los que están situados en la parte superior de la tabla, por lo tanto se puede determinar que dicho método realiza su función correctamente.

G. Pruebas, entorno con un grupo pequeño de usuarios, resultados de la encuesta.

En las siguientes ilustraciones se muestran los resultados obtenidos tras realizar una prueba del recomendador desarrollado en este trabajo de fin de grado con un pequeño grupo de usuarios. Las condiciones de la prueba se han definido en la ***sección 6 "Resultados"***.

- **Pregunta 1. ¿Con que frecuencia accedes a contenidos informativos de actualidad? Tanto prensa escrita como Internet, radio o televisión.**

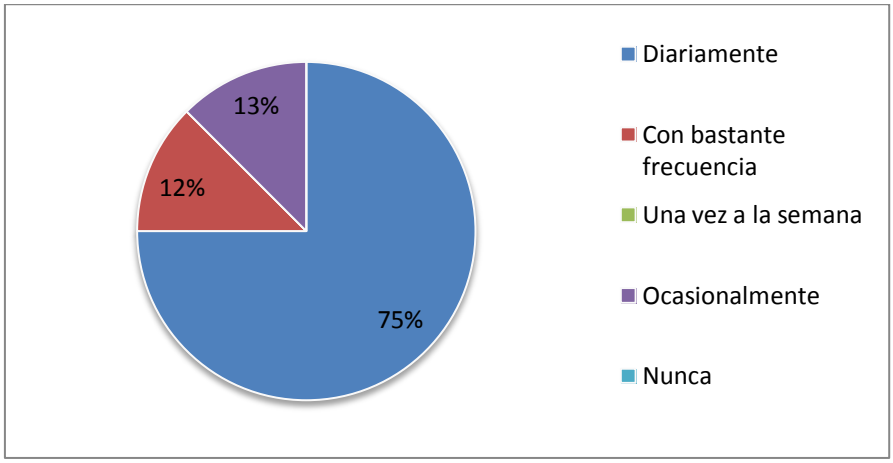

Ilustración 36. Encuesta, pregunta 1.

- **Pregunta 2. ¿Cuál es la fuente que utilizas habitualmente como canal de comunicación?**

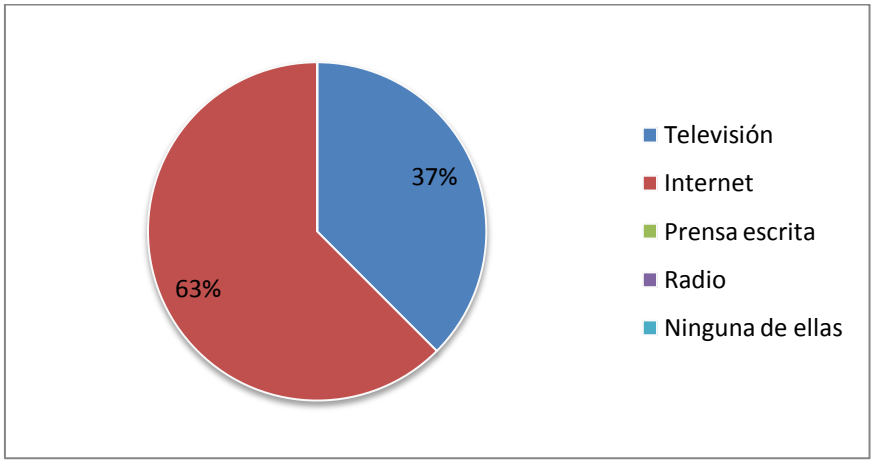

Ilustración 37. Encuesta, pregunta 2.

- **Pregunta 3. ¿Te han resultado interesantes las noticias mostradas?**

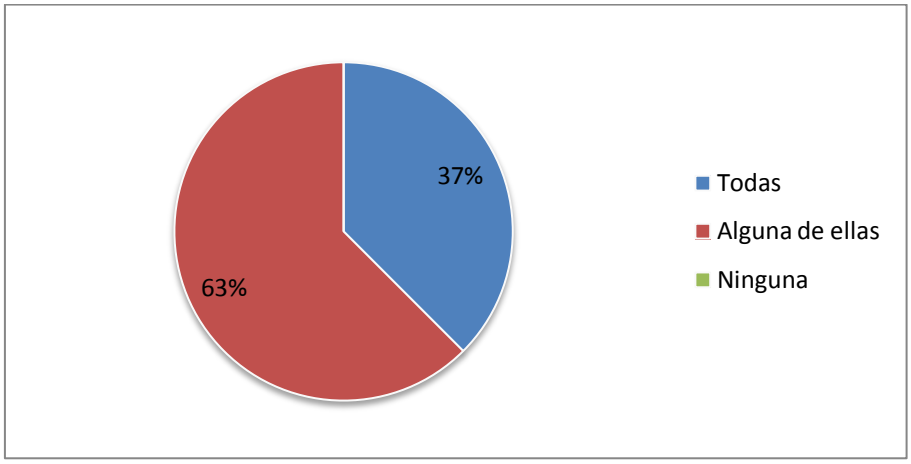

Ilustración 38. Encuesta, pregunta 3.

- **Pregunta 4. ¿Por qué crees que esas noticias te han resultado interesantes?**

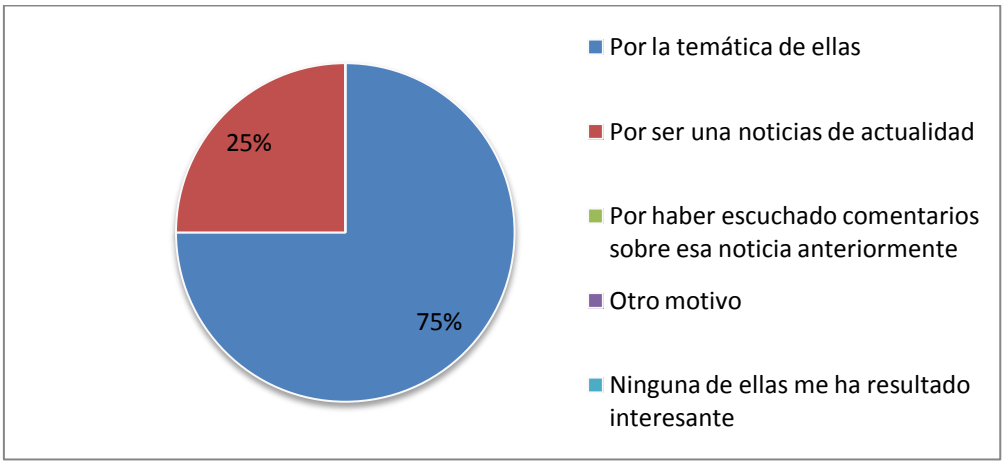

Ilustración 39. Encuesta, pregunta 4.

- **Pregunta 5. Sobre las noticias que te han resultado interesantes, ¿Crees que hubieras accedido a ellas simplemente viendo su entradilla?**

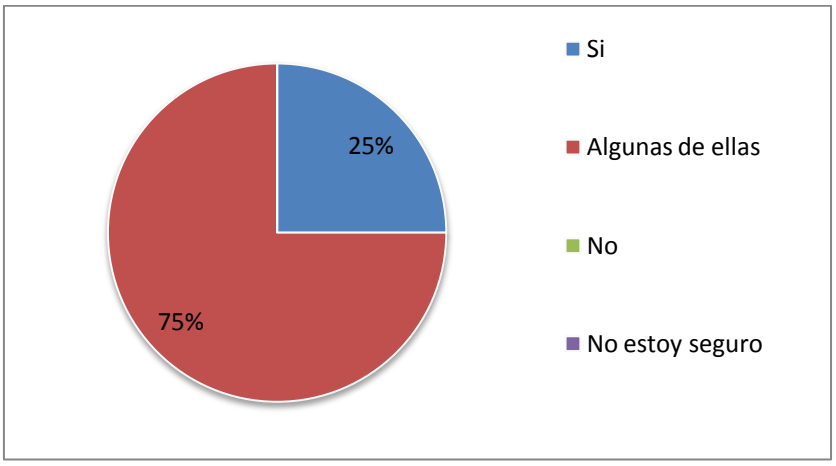

Ilustración 40. Encuesta, pregunta 5.

- **Pregunta 6. ¿Por qué crees que esas noticias no te han resultado interesantes?**

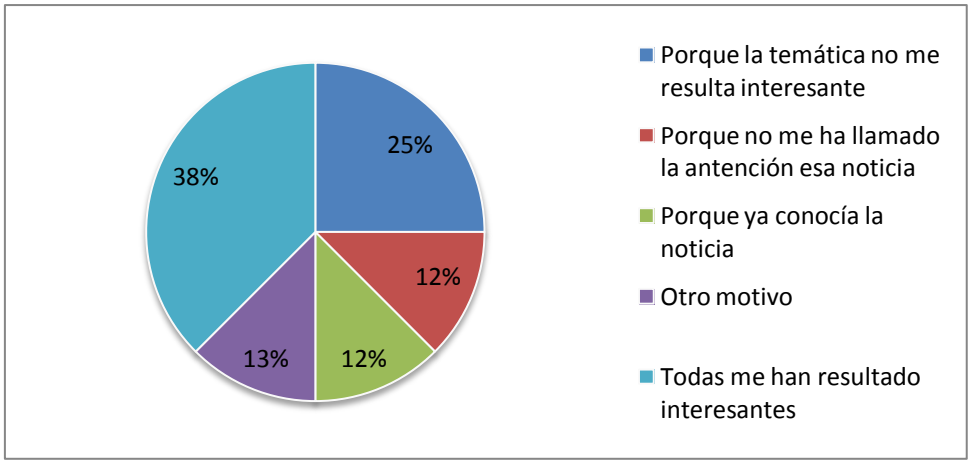

Ilustración 41. Encuesta, pregunta 6.

- **Pregunta 7. Observando las noticias disponibles en la web del publicador, ¿Te hubiera resultado más interesante otra selección de noticias distinta a la recomendada?**

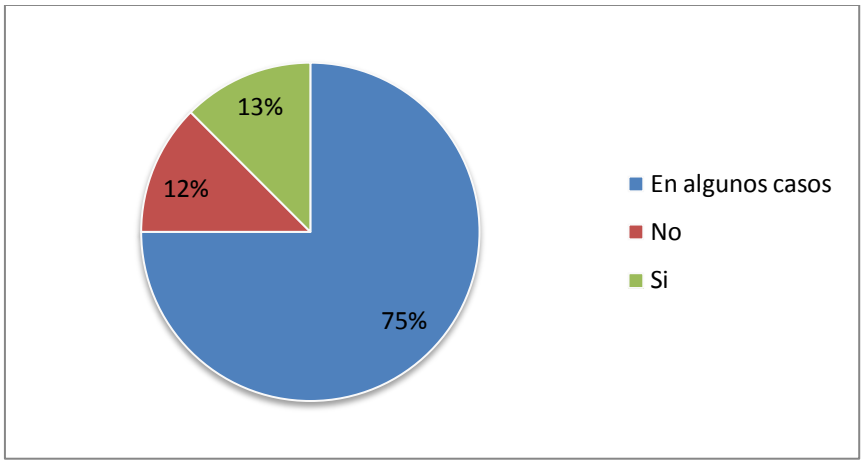

Ilustración 42. Encuesta, pregunta 7.

- **Pregunta 8. En cuanto a las noticias recomendadas, ¿Cuál crees que es el elemento que más condiciona para visitar alguna de esas noticias?**

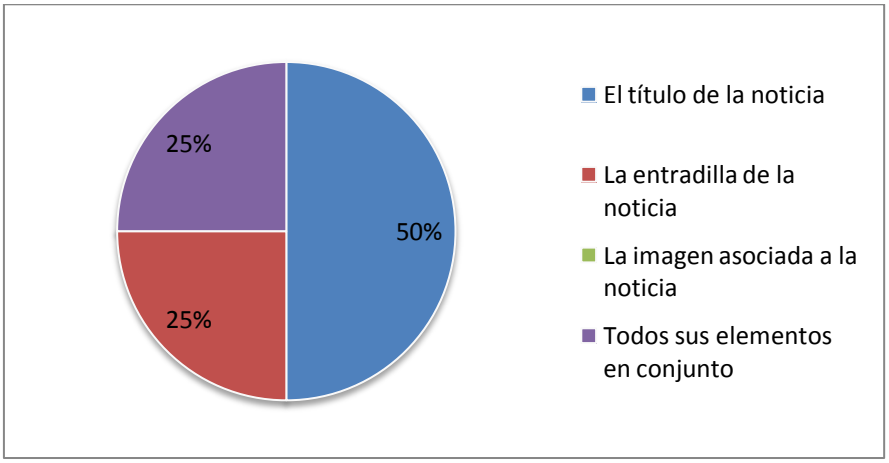

Ilustración 43. Encuesta, pregunta 8.